UNIVERSITY OF CALIFORNIA

Los Angeles

Time Synchronization in Wireless Sensor Networks

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Jeremy Eric Elson

2003

Report Documentation Page

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **2003**	2. REPORT TYPE	3. DATES COVERED **00-00-2003 to 00-00-2003**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Time Synchronization in Wireless Sensor Networks	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California, Los Angeles,Computer Science Department,Electrical Engineering Department,Los Angeles,CA,90095**	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**202**	

The dissertation of Jeremy Eric Elson is approved.

Mario Gerla

Gerald J. Popek

Gregory J. Pottie

Majid Sarrafzadeh

Deborah L. Estrin, Committee Chair

University of California, Los Angeles

2003

For Grandma

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I have received such immeasurable support from so many people, and in such diverse ways, that it seems an impossible task to enumerate them.

My graduate career started out at USC, which I visited as a potential student in April of 1998. One of the first people I met was Reza Rejaie, then a soon-to-graduate Ph.D. student himself. During a tour of the networking lab, he took me aside and said: "You should come to school here. You might find better facilities or a nicer location, but you can search the world and not find an advisor better than Deborah."

In the years that followed, I learned how true his words were. My greatest and heartfelt thanks must go first to Deborah Estrin, my thesis advisor, mentor, and friend. For her students, she unendingly gives of herself, while being unfailingly selfless. Deborah has incredible vision and boundless energy. She teaches research discipline by example, and shows all of her students how to start from an ultimate vision of the world and reduce it to a tractable problem. Beneath the researcher is a gentle soul, ever forgiving, encouraging, healing, and inspiring. It is hard to imagine having done my Ph.D. with anyone else.

Many other faculty members also deserve my sincere thanks. John Heidemann is a spectacular role-model as a systems researcher; working with him for the past few years have been inspiring. Greg Pottie is brilliant communications theorist and has an uncanny knack for cutting to the core of any problem, separating the truly relevant wheat from the obfuscating chaff. I have enjoyed working with Mani Srivastava and Kung Yao, whose labs have worked with us even before the creation of CENS. I'd also like to thank the members of my qualifying and thesis committees: Mario Gerla, Gerry Popek, Greg Pottie, and Majid Sarrafzadeh.

Each of them provided me with invaluable feedback and encouragement. In his role as my OSDI shepherd, Eric Brewer provided insightful comments that significantly improved on our original RBS paper. Ted Faber entrusted his operating systems class to me in 1998—my first job after I arrived at USC.

Bill Kaiser deserves special mention: he has been ever supportive of my work, both as a co-founder and CTO of Sensoria Corporation and a professor at UCLA. Bill can effortlessly move among many roles—a grand visionary describing plans for years into the future, and a pragmatist engineer attuned to the details needed to meet deadlines tomorrow. My position at Sensoria allowed me to pursue my doctoral degree while simultaneously applying my ideas to projects in the real world. I believe both my degree and Sensoria were able to benefit mutually, and I thank Bill for having the flexibility to allow this arrangement. I would also like to thank the other members of Sensoria's research group—in particular, Lew Girod, Billy Merrill, Fredric Newberg, and Kathy Sohrabi.

Lew Girod was also my first graduate-school officemate; as the years passed, he has become one of my most valued and trusted collaborators and friends. Lew and I have worked together extensively at both UCLA and Sensoria; both my research and my *approach* to research have benefitted greatly as a result.

Many other students also helped make graduate school a better place both through research collaboration and friendship; I am grateful for both. Alberto Cerpa[1] has worked closely with me for years on projects ranging from NECP to EmStar. Vlad Bychkovskiy and Thanos Stathopoulos have been a great help in unlocking the secrets of TinyOS. Naim Busek and Mohammed Rahimi have infinite patience for endless questions from a computer scientist who must occasionally play the role of an electrical engineer. Nirupama Bulusu served as the

---

[1]We rule.

archetype of a LECS graduate, lighting a path that was often murky. I'd also like to thank Solomon Bien, Deepak Ganesan, Ben Greenstein, Tom Schoellhammer, Fabio Silva, Hanbiao Wang, and Yan Yu.

If one assumes I have not gone insane, credit for this may go to my family, who have provided endless emotional support. In particular, I'd like to thank my parents Norton and Sandy, and siblings Franny and David, for tirelessly being there through success and failure. I must thank Dana for giving me a great reason to be in Santa Monica by 6:17 every Friday, while having patience and understanding during the times that I can't make it. These people are so close to my heart, and have given me such strength and encouragement, that "thank" seems too cold and empty of a word.

I'd also like to thank Gary Klapow—his patient instruction has made Michael Althsuler's words (Chapter 10) a literal reality for me. At a time when I felt that my life had been consumed by digital work, he helped me enjoy an old hobby in analog rather than digital form.

Finally, I should of course thank Reza, without whom I might have gone down a completely different path. This one has worked out pretty well.

Jeremy Elson   *Santa Monica, California, May 2003*

Vita

1974	Born, San Francisco, California, USA.
1991	Graduated, Walt Whitman High School, Bethesda, Maryland.
1996	B.S., Computer Science, Johns Hopkins University, Baltimore, Maryland.
1996–1998	Computer Engineer, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Maryland. Lead software engineer in the development of a new ATM-based telemedicine workstation.
1998–1999	Graduate Teaching Assistant, Department of Computer Science, University of Southern California (USC), Los Angeles, California. Taught two semesters of the undergraduate course Introduction to Operating Systems.
1999–2000	Graduate Research Assistant, USC/Information Sciences Institute, Computer Networks Division, Marina del Rey, California.
1999–2002	Protocol Design Consultant, Network Appliance, Sunnyvale, California. Designed and authored specifications for "NECP," the Network Element Control Protocol, and "ICAP," the Internet Content Adaptation Protocol.
2000	M.S., Computer Science, University of Southern California (USC), Los Angeles, California.

| 2000–2002 | Senior Design Engineer, Sensoria Corporation, Culver City, California. Designed and implemented software for networked embedded systems; applied thesis research (e.g., time synchronization, acoustic localization, network self-assembly) to a variety of deployed autonomous sensor-network projects. |
| 2000–2003 | Graduate Research Assistant, Department of Computer Science, Laboratory for Embedded Collaborative Systems (LECS), University of California, Los Angeles (UCLA), Los Angeles, California. |

PUBLICATIONS

Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology. In *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, 3-5 April, 2001, San Jose, Costa Rica.

Jeremy Elson and Deborah Estrin. Random, Ephemeral Transaction Identifiers in Dynamic Sensor Networks. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, April 16-19, 2001, Phoenix, Arizona, USA.

Jeremy Elson and Deborah Estrin. Time Synchronization for Wireless Sensor Networks. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS)*, Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, April 2001, San Francisco, CA, USA.

Jeremy Elson, Lewis Girod, and Deborah Estrin. Short Paper: A Wireless Time-Synchronized COTS Sensor Platform, Part I: System Architecture. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, California. September 2002.

Jeremy Elson and Kay Römer. Wireless Sensor Networks: A New Regime for Time Synchronization. In *Proceedings of the First Workshop on Hot Topics In Networks (HotNets-I)*, Princeton, New Jersey. October 2002.

Jeremy Elson, Lewis Girod and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA. December 2002.

Jeremy Elson, Solomon Bien, Naim Busek, Vladimir Bychkovskiy, Alberto Cerpa, Deepak Ganesan, Lewis Girod, Ben Greenstein, Tom Schoellhammer, Thanos Stathopoulos, and Deborah Estrin. EmStar: An Environment for Developing Wireless Embedded Systems Software. CENS Technical Report 0009, March 24, 2003.

Jeremy Elson and Alberto Cerpa, editors. ICAP: The Internet Content Adaptation Protocol. RFC 3507. April 2003.

Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin. Locating tiny sensors in time and space: A case study. In *Proceedings of the International Conference on Computer Design (ICCD 2002)*, Freiburg, Germany. September 2002. Invited paper.

John Heidemann, Nirupama Bulusu, Jeremy Elson, Chalermek Intanagonwiwat, Kun-chan Lan, Ya Xu, Wei Ye, Deborah Estrin, and Ramesh Govindan. Effects of Detail in Wireless Network Simulation In *Proceedings of the SCS Multiconference on Distributed Simulation,* Phoenix, Arizona, USA, Society for Computer Simulation. January, 2001.

Kenneth M. Kempner, David Chow, Peter Choyke, Jerome R. Cox, Jr., Jeremy E. Elson, Calvin A. Johnson, Paul Okunieff, Harold Ostrow, John C. Pfeifer, and Robert L. Martino. The development of an ATM–based Radiology Consultation WorkStation for radiotherapy treatment planning. In *Image Display: Medical Imaging 1997 Conference Proceedings*, volume 3031, pp. 500–511. Society of Photo-Optical Instrumentation Engineers, 1997.

William Merrill, Lewis Girod, Jeremy Elson, Kathy Sohrabi, Fredric Newberg, and William Kaiser. Autonomous Position Location in Distributed, Embedded, Wireless Systems. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, California. September 2002.

Katayoun Sohrabi, William Merrill, Jeremy Elson, Lewis Girod, Fredric Newberg, and William Kaiser. Scaleable Self-Assembly for Ad Hoc Wireless Sensor Networks. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, California. September 2002.

Hanbiao Wang, Len Yip, Daniela Maniezzo, Joe C. Chen, Ralph E. Hudson, Jeremy Elson, and Kung Yao. A Wireless Time-Synchronized COTS Sensor Platform, Part II: Applications to Beamforming. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, California. September 2002.

Hanbiao Wang, Jeremy Elson, Lewis Girod, Deborah Estrin, and Kung Yao. Target Classification and Localization in Habitat Monitoring. To appear in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2003)*, Hong Kong, China. April 2003.

<div align="center">

ABSTRACT OF THE DISSERTATION

Time Synchronization in Wireless Sensor Networks

by

Jeremy Eric Elson

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2003

Professor Deborah L. Estrin, Chair

</div>

Recent advances in miniaturization and low-cost, low-power design have led to active research in large-scale networks of small, wireless, low-power sensors and actuators. Time synchronization is a critical piece of infrastructure in any distributed system, but wireless sensor networks make particularly extensive use of synchronized time. Almost any form of sensor data fusion or coordinated actuation requires synchronized physical time for reasoning about events in the physical world. However, while the clock accuracy and precision requirements are often stricter in sensor networks than in traditional distributed systems, energy and channel constraints limit the resources available to meet these goals.

New approaches to time synchronization can better support the broad range of application requirements seen in sensor networks, while meeting the unique resource constraints found in such systems. We first describe the design principles we have found useful in this problem space: *tiered* and *multi-modal* architectures are a better fit than a single solution forced to solve all problems; *tunable* methods allow synchronization to be more finely tailored to problem at hand; *peer-to-peer*

synchronization eliminates the problems associated with maintaining a global timescale. We propose a new service model for time synchronization that provides a much more natural expression of these techniques: *explicit timestamp conversions.*

We describe the implementation and characterization of several synchronization methods that exemplify our design principles. *Reference-Broadcast Synchronization* achieves high precision at low energy cost by leveraging the broadcast property inherent to wireless communication. A novel multi-hop algorithm allows RBS timescales to be federated across broadcast domains. *Post-Facto Synchronization* can make systems significantly more efficient by relaxing the traditional constraint that clocks must be kept in continuous synchrony.

Finally, we describe our experience in applying our new methods to the implementation of a number of research and commercial sensor network applications.

CHAPTER 1

Introduction

It was the best of times, it was the worst of times...

—Charles Dickens, A Tale of Two Cities

1.1 Wireless Sensor Networks

Recent advances in miniaturization and low-cost, low-power design have led to active research in large-scale, highly distributed systems of small, wireless, low-power, unattended sensors and actuators [ASS02, ADL98, KKP99, EGH99]. The vision of many researchers is to create sensor-rich "smart environments" through large-scale deployment of microprocessors into the environment, each combined with radios capable of short-range wireless communication and sensors that can detect local conditions such as temperature, sound, light, or the movement of chemicals or objects. There are a wide range of applications envisioned for such sensor networks, including microclimate studies [CEE01, MPS02], groundwater contaminant monitoring [KPH03], precision agriculture, condition-based maintenance of machinery in complex environments, urban disaster prevention [BCY98] and response, and military interests [SHM, NES].

Such applications are not well served by traditional sensing architectures. Small numbers of high-powered, long-range sensors have traditionally worked well

in unobstructed environments—for example, sky observations by weather radar—but are not effective in these complex, cluttered environments where line-of-sight paths are typically very short. The clutter effect can be reduced by distributing larger numbers of shorter-range sensors closer to their targets. Such placement also improves the signal-to-noise ratio, critical for sensing phenomena that are transmitted with high path loss, such as magnetic fields and high-frequency seismic motion. However, in many contexts, it is prohibitively impractical to deploy large numbers of *wired* sensors in large areas, as they require co-deployed infrastructure (power, communications). Manual observation is a time-honored tradition in fields such as environmental monitoring, but it is labor-intensive—to cover a large area, observation must be very low-frequency, both temporally and spatially. Thus, it fails to capture the events (signals) of interest, which often occur at higher frequencies.

Ad-hoc deployable, wireless sensor networks can observe the environment in a fundamentally different way than previous classes of systems—close to the phenomena in question, over a wide area, and densely in both time and space. They may succeed in applications where traditional solutions have failed. This vision has captured the interest and imagination of many scientists; as Prof. Tom Harmon has noted, sensor networks hold the promise of *revealing previously unobservable phenomena* in the environment. Eventually, such systems will also be capable of actuation—affecting the environment rather than solely observing it [BCY98, SRS02, SHM].

The design of such systems poses serious challenges, and has become the focus of a broad range of active research. Sensor networks differ substantially in many ways from traditional distributed systems. These differences often violate assumptions on which many of the traditional techniques are predicated. Sen-

sor networks have become a fertile ground for new designs that take the new assumptions into account.

Perhaps the most important characteristic of sensor networks—and the one that differentiates them most dramatically from traditional distributed systems—is the crucial need for *energy efficiency*. To facilitate easy deployment without an infrastructure, many nodes will necessarily be *untethered*, having only finite energy reserves from a battery. Unlike laptops or other handheld devices that enjoy constant attention and maintenance by humans, the scale of a sensor network's deployment will make replenishment of these reserves impossible. This requirement pervades all aspects of the system's design, and drives most of the other requirements.

Fundamental physical limits dictate that, as electronics become ever more efficient, *communication* will dominate a node's energy consumption [PK00]. The disproportionate energy cost of long-range vs. short-range transmission (r^2 to r^4), as well as the need for spatial frequency reuse, precludes communication beyond a short distance. Therefore, a common research theme has been in maximizing local processing and minimizing the overhead of collaboration. Emerging designs allow users to task the network with a high-level query such as "notify me when a large region experiences a temperature over 100 degrees" or "report the location where the following bird call is heard" [HSI01, MFH02]. If a node can correlate an incoming audio stream to the desired pattern *locally*, and report only the time and location of a match, the system will be many orders of magnitude more efficient than one that transmits the complete time-series of sampled audio.

The use of local processing, hierarchical collaboration, and domain knowledge to convert data into increasingly distilled and high-level representations—or, *data reduction*—is key to the energy efficiency of the system. In general, a perfect

system will reduce as much data as possible as early as possible, rather than incur the energy expense of transmitting raw sensor values further along the path to the user.

Another fundamental property of sensor networks is their *dynamics*. Over time, nodes can fail—they may run out of energy, overheat in the sun, be carried away by wind, crash due to software bugs, or be eaten by a wild boar. Even in fixed positions, nodes' communication ranges (and, thus, topologies) can change dramatically due to the vagaries of RF propagation, a result of its strong environmental dependence. These changes are difficult to predict in advance. Traditional large-scale networks such as the Internet work in the face of changing configurations and brittle software partly because the number of people maintaining the network has grown along with the size of the network itself. In contrast, there may be a single human responsible for thousands of nodes in a dense sensor network. Any design in which each device requires individual attention is infeasible. This leads to another important requirement: sensor networks must be *self-configuring*, and *adaptive* to changes in their environment.

The unique design requirements in sensor networks affect virtually every aspect of a system's design—routing and addressing mechanisms, naming and binding services, application architectures, security mechanisms, and so forth. Indeed, even the development process itself—the iterative process of a system's design, construction, and evaluation—has required adaptation to this new regime [EBB03].

1.2 Time Synchronization in Sensor Networks

Time synchronization is a critical piece of infrastructure in any distributed system. In sensor networks, a confluence of factors makes flexible and robust time synchronization particularly important, while simultaneously making it more difficult to achieve than in traditional networks.

Collaboration among nodes is often required for the data reduction that is critical to the energy-efficiency of a sensor network. A common view of physical time is a basic requirement for nodes to reason about events that occur in the physical world. For example, precise time is needed to measure the time-of-flight of sound [GE01, MGE02]; distribute an acoustic beamforming array [WYM02]; form a low-power TDMA radio schedule [ADL98]; integrate a time-series of proximity detections into a velocity estimate [CEE01]; or suppress redundant messages by recognizing duplicate detections of the same event by different sensors [IGE00]. In addition to these domain-specific requirements, sensor network applications often rely on synchronization as typical distributed systems do: for database queries [BGS00, MF02], cryptography and authentication schemes [PSW01, EG02, LEH03], coordination of future action, interaction with users, ordering logged events during system debugging, and so forth.

The many uses of synchronized time in a sensor network make it critical. However, the diversity of these roles also makes synchronization a difficult problem to solve. Application requirements vary widely on many axes, such as precision, lifetime, scope, availability, and energy budget. For example, acoustic applications require precision of several microseconds, while sensor tasking works on the timescale of hours or days. Local collaborations often require only a pair of neighbors to be synchronized, while global queries require global time. Event triggers may only require momentary synchronization, while data logging or debugging

5

often require an eternally persistent timescale. Communication with a user requires a external, human timescale such as UTC, whereas only *relative* time is important for purely in-network comparisons. Some nodes have large batteries and run all the time; others are so constrained that they only wake up occasionally, take a single sensor reading, and transmit it before immediately returning to sleep.

Even in a traditional distributed system, creation of a synchronization scheme that satisfies such a broad spectrum of requirements is challenging. The task becomes particularly daunting in sensor networks, in light of their additional domain requirements—including energy efficiency, scalability through localized interactions, and automatic adaptation to dynamics. For example, the energy constraints violate a number of assumptions routinely made by classical synchronization algorithms: that using the CPU in moderation is free, listening to the network is free, and occasional transmissions have a negligible impact. Network and node dynamics require continuous, automatic configuration; this precludes *a priori* selection of a particular node as the "master clock." Indeed, as we will argue in Chapter 5, the scale of the network and intermittency of its links may preclude the very existence of any *single* master clock node.

A paradox of sensor networks, then, is that they make stronger demands on a time synchronization system than traditional distributed systems, while simultaneously limiting the resources available to achieve it. This paradox has made current synchronization schemes inadequate to the task.

1.3 Contributions

Computer clock synchronization is a well-studied problem, with a rich history stretching back to the advent of the first computer networks [Mil03]. However, the unique system architecture seen in sensor networks brings a new dimension to many old problems. Our contributions fall into several categories:

Definition of the problem. We consider the various uses of time synchronization in detail, and describe the axes along which these applications can be characterized. A detailed understanding of the requirements is important in this realm where efficiency is so critical.

New architecture directions. Based on our experience exploring this problem space, we propose several general guidelines for the design of time synchronization systems in sensor networks. Specifically, we propose new systems be:

- *Multi-modal*—No *single* synchronization scheme can be optimal on all axes (e.g., precision, lifetime, scope, energy, availability), but most applications do not require peak performance on all axes. A range of schemes should be available to system designers, such that the synchronization that is provided matches what is needed.

- *Tunable*—An ideal synchronization system will minimize its energy use by providing service that is exactly necessary and sufficient for the needs of the application. Tunable parameters can allow synchronization modes to be matched more closely to the requirements of the application.

- *Tiered*—Although Moore's law predicts that hardware for sensor networks will inexorably become smaller, cheaper, and more powerful, technological advances will never prevent the need to make trade-offs. Instead of choosing

7

a single hardware platform that makes a particular set of compromises, we believe an effective design is one that uses a tiered platform consisting of a heterogeneous collection of hardware.

- *Explicit*—Most existing time synchronization schemes make a common assumption: that their goal is to keep the clock synchronized all the time. Applications assume that they can query the clock at any time, and it will be synchronized. We argue that in sensor networks, synchronization should *not* attempt to discipline the phase and frequency of oscillators. A better approach is to let clocks run at their natural rate, and use synchronization to build a set of relationships to other clocks or synthetic timescales. Applications can query the local clock, then perform an *explicit* conversion to another timescale. This has many advantages; for example, it enables post-facto synchronization, peer-to-peer synchronization, and participation in multiple timescales.

- *Peer-to-peer*—Traditional synchronization schemes assume the existence of a *global timescale*, usually realized by a *master clock*. We argue that the goal of a synchronization scheme should, instead, be to define the relationship between nearby clocks in a *peer-to-peer* fashion, without necessarily constructing a global timescale. This has a number of benefits—primarily, error between nodes is proportional to the distance between them, rather than their distance from a master clock that is likely to be far away.

Design, implementation, and characterization of new methods. We contribute three new methods to the palette of synchronization schemes available to sensor network developers:

- *Reference-Broadcast Synchronization*—A form of time synchronization that leverages the broadcast property inherent to wireless communication. A reference broadcast does not contain an explicit timestamp; instead, receivers use its arrival time as a point of reference for comparing their clocks. We use measurements from two wireless implementations to show that removing the sender's nondeterminism from the critical path in this way produces high-precision clock agreement ($1.85 \pm 1.28\mu$sec, using off-the-shelf 802.11 wireless Ethernet), while using minimal energy.

- *Multi-hop RBS*—A novel algorithm that federates clocks across broadcast domains, without losing the critical receiver-to-receiver property of RBS. Our analysis shows precision should decay slowly; the standard deviation grows as $O(\sqrt{n})$ after n hops. In our implementation, we measured an error of $3.68\pm2.57\mu$sec after 4 hops. The multi-hop algorithm allows coordination of any size group; the error incurred is only as large as the distance between the nodes using the synchronization.

- *Post-Facto Synchronization*—A technique that allows significant energy savings in networks where time synchronization is needed occasionally and unpredictably. Instead of keeping clocks synchronized all the time, clocks run undisciplined (at their natural rate), and are reconciled after an event of interest occurs. We show experimental results that explore the error incurred with variations on this scheme.

Application experience. We describe a number of systems in which we have applied our time synchronization techniques to the construction of distributed sensor network applications. Most of these systems use RBS-synchronized clocks to precisely measure the arrival time of acoustic signals. Audio codecs typically

9

sample the channel at 48KHz, or $\approx 21\mu$sec per sample. As we will see in Chapter 6, RBS nominally achieves significantly better precision. Distributed acoustic sensors have thus been able to correlate their acoustic time-series down to individual samples. We show how this was crucial for the implementation of distributed acoustic ranging and blind beamforming and source localization of sound. We also describe a commercial system that uses RBS for acoustic ranging.

1.4   Dissertation Overview

In the remainder of this dissertation, we will consider time synchronization in sensor networks in more detail: its motivations, the inherent limitations, and our new solutions.

Chapter 2 reviews related work in computer clock synchronization, and the measurement of time and frequency.

Chapter 3 describes how synchronized time is a critical service in sensor networks—it is a basic requirement for virtually all algorithms that reason about time-varying observations made by distributed sensors. By examining several typical applications and their synchronization requirements, we can more precisely define the problem that needs to be solved.

In Chapter 4, we explore a number of metrics that we have found relevant for evaluating time synchronization in the sensor network domain. These metrics are useful both for decomposing the synchronization required by an application, and understanding what is provided by a particular synchronization method.

In Chapter 5, we present a more detailed argument that the new constraints seen in sensor networks make traditional time synchronization methods ill-suited for serving the diverse range of sensor network application requirements. We also

describe general design principles for sensor network time synchronization that we have found useful in the course of work: multi-modal synchronization, peer-to-peer synchronization without a global timescale, tiered and tunable modes, and integration with the application.

Chapter 6 describes *Reference-Broadcast Synchronization* in detail, building up from its fundamental ideas to a description of its implementation and performance characterization. This section also describes the first two platforms on which RBS was implemented. The first is the Berkeley Mote, using TinyOS [HSW00] and a narrowband low-bit-rate RF Monolithics radio. The second is a commercial-off-the-shelf platform based around a Compaq iPAQ PDA running Familiar Linux [FAM] and spread-spectrum 802.11b wireless Ethernet.

In Chapter 7, we present our novel algorithm that allows clocks to be federated across broadcast domains, without losing the critical receiver-to-receiver property of RBS. We show an analysis of the scheme's expected error, and test it empirically using the 802.11-based wireless testbed described in Chapter 6.

In Chapter 8, we describe *Post-Facto Synchronization*, a technique that allows significant energy savings in networks where time synchronization is needed occasionally and unpredictably. We show experimental results that explore the error incurred with variations on this scheme.

Chapter 9 relates our experience building a number of sensor networks systems that use our time synchronization techniques. Our implementations show the viability of our design principles (multi-modal and peer-to-peer synchronization) and our methods (RBS, multi-hop RBS, and post-facto synchronization).

Finally, in Chapter 10, we summarize our work and present our conclusions. We also describe directions for future research in this area.

CHAPTER 2

Related Work

Time is God's way of keeping everything from happening at once.

—Unknown

Computer clock synchronization is a well-studied problem, with a rich history stretching back to the advent of the first computer networks [Mil03]. The science of timekeeping itself has been a preoccupation of every civilization since the dawn of history [NPL02, Lan83]. Of course, we can not review the complete body of that work in detail here. In this chapter, we will consider selected work that is particularly relevant to our research, or which we have found especially important or inspirational.

We begin by sampling some of the extensive literature on computer clock synchronization in traditional networks in Section 2.1. Section 2.2 reviews the recent work from other groups who, like us, have extended network time synchronization into the sensor network domain. In Section 2.3, we describe a number of projects that have found ways to achieve tight synchronization by leveraging specialized hardware.

The first three sections describe ways to synchronize clocks. In Section 2.4, we describe somewhat more formally just what a clock *is*, and how the properties of clocks themselves can interact with methods for synchronizing them. The definition of a *second* is discussed in Section 2.5; in Section 2.6, we describe how

national laboratories use that definition to construct and disseminate persistent timescales such as UTC.

Finally, in Section 2.7, we describe a slightly different approach time synchronization: designing a system such that the problem does not need to be solved.

2.1 Network Time Protocols

Over the years, many protocols have been designed for maintaining synchronization of physical clocks over computer networks [Cri89, GZ89, Mil94a, ST87]. These protocols all have basic features in common: a simple connectionless messaging protocol; exchange of clock information among clients and one or more servers; methods for mitigating the effects of nondeterminism in message delivery and processing; and an algorithm on the client for updating local clocks based on information received from a server. They do differ in certain details: whether the network is kept internally consistent or synchronized to an external standard; whether the server is considered to be the canonical clock, or merely an arbiter of client clocks, and so on.

There is extensive literature describing variations on these methods. Most of it conforms to the basic model of a synchronization protocol described above and proposes statistical and heuristic methods for improving robustness: better outlier rejection, reduction of the effects of jitter (non-determinism), synchronization in the face of Byzantine failures, and so forth. For example:

- Cristian has observed that performing larger numbers of request/response experiments will make it more likely that at least one trial will not encounter random delays [Cri89]. This trial, if it occurs, is easily identifiable as the shortest round-trip time.

- Marzullo and Owicki have described an algorithm that produces a correct estimate of the current time, given the information from n time servers, provided that fewer than $n/2$ of the servers have failed [MO85].

- Mills has developed a control loop for disciplining the frequency of a local oscillator so that it eventually matches an external time reference in both phase and frequency [Mil98].

Mills' Network Time Protocol, or NTP [Mil94a], stands out by virtue of its scalability, self-configuration in large multi-hop networks, robustness to failures and sabotage, and ubiquitous deployment. NTP allows construction of a hierarchy of time servers, multiply rooted at canonical sources of external time. The protocol has evolved considerably over its three decades of continuous use, and has gradually been updated to incorporate many of the new ideas from the research community (including the three described above). NTP is now the *de facto* standard for time synchronization on the Internet; we consider it the "gold standard" by which other work should be judged.

Perhaps most closely related to our RBS protocol (described in Chapter 6) are the CesiumSpray system [VRC97], the foundations of which were described by Veríssimo and Rodrigues [VR92]; and the 802.11-based broadcast synchronization described by Mock *et al.* [MFN00]. Their systems, like ours, make use of the inherent properties of broadcast media to achieve superior precision. However, their methods require all nodes to lie within a single broadcast domain; multiple domains can not be federated, except by depending on an out of band common view of time (e.g., GPS, as described in Section 2.6). In contrast, RBS incorporates a novel multi-hop algorithm; in Chapter 7, we show it can maintain tight time synchronization across many hops through a wireless network without external infrastructure support. In addition, we have fully decoupled the sender

from the receiver—only synchronizing receivers with one another, even when relating the timescale to an external timescale such as UTC. CesiumSpray retains a dependence on the relationship between sender and receiver when synchronizing nodes to UTC.

2.2 Time Synchronization for Sensor Networks

A number of other groups have considered synchronized time in sensor networks, concurrently with our work.

Römer suggests that, for some applications, the time of events can be described in terms of their *age* rather than as an absolute time [R01]. In essence, when two nodes exchange a message that describes an event in terms of is age, the time at which the message itself is sent becomes a common frame of reference for time. The notion of "now" at the instant a message is sent is inexact, due to nondeterministic and asymmetric latencies; this is an important source of error. In the long term, error is dominated by frequency differences among nodes; the effect is magnified as timestamps age. Römer's analysis of his scheme concludes it has a nominal precision of 1ms.

Su and Akyildiz describe the Time Diffusion Protocol, or TDP, for achieving global time synchronization in sensor networks [SA02]. Their work has a number of strengths, including a automatic self-configuration through dynamic topology discovery. In addition, they quantitatively analyze the energy cost of their scheme. However, to date, TDP has not been implemented, and the authors' simulations leave out certain details—such as a realistic model of the channel's non-determinism—that are critical to understanding its expected performance.

The Berkeley Mote [KKP99], using the TinyOS operating system [HSW00],

has become a popular development environment for sensor network research. TinyOS allows complete control over the network stack, making it possible to integrate application code all the way to the MAC layer, and build timing-aware, low-jitter bit detectors into the radio stack. Hill *et al.* predict that these methods can achieve 2μsec precision for receivers within a single broadcast domain [HC01].

Ganeriwal *et al.*, also using TinyOS, have implemented an algorithm that automatically constructs an NTP-like hierarchy of time-synchronized nodes. Similar to TDP, their scheme automatically elects a node as the canonical source of time, and is robust to topology changes. As in Hill's scheme, their implementation reduces jitter by tightly integrating the time synchronization algorithm with the MAC; they report 25μsec-per-hop precision across multiple hops [GKA02].

Our RBS scheme (Chapter 6) does not require that the application be collapsed into the MAC. This allows RBS to work over a broader class of network technologies, including off-the-shelf radios where the MAC can not be modified. In Section 6.4, we will describe an RBS implementation on commodity hardware—802.11 wireless Ethernet.

However, our methods can benefit from low-jitter radio stacks such as those created by Hill and Ganeriwal, and we consider such work as being complementary to ours. Our focus has not been on building deterministic networks or bit-detectors, but rather making the best use of existing detectors. That is, RBS is a technique to achieve the best possible synchronization given a *particular* bit detector. Better (i.e., more deterministic) bit detectors will lead to better RBS performance. For example, by leveraging Hill's 0.25μsec-precision bit detector, RBS could achieve several times the precision of their scheme, based on their 2μsec jitter budget analysis.

2.3 Support from Deterministic Hardware

The enemy of precise network time synchronization is *non-determinism*. Latency estimates are confounded by random events that lead to asymmetric round-trip message delivery delays; this contributes directly to synchronization error. Purely software-based schemes are often designed to run in asynchronous environments that do not have tight timing guarantees—networks like Ethernet, or in the extreme, the Internet. In addition, software often runs on a general purpose operating system, where there are many opportunities for non-determinism to creep in: the connection between the network interface and the host CPU, process scheduling policies of the operating system, and so forth.

Consequently, a common way to reduce jitter in message delivery is to construct special-purpose hardware that provides tight timing guarantees, or to eliminate software scheduling delay by putting software hooks as close to the "bare metal" as possible. Some examples of this technique include the following:

- Liao *et al.* describe a system for synchronizing interfaces on a system-area network, Myrinet, with microsecond accuracy [LMC99]. Myrinet allows user code to be inserted into the network interface, executed synchronously as messages arrive. This provides tight timing bounds on message delivery, known *a priori*, if the topology is fixed.

- Abali *et al.* describe clock synchronization on the IBM SP/2 multicomputer [ASB96]. By leveraging the special-purpose node interconnection hardware unique to the SP/2, they achieved clock agreement within a cluster to within 25 to 200 nanoseconds, depending on the size of the network. Their method also reckons the cluster time to an external timescale (e.g., UTC served via NTP) to within several microseconds.

- Mills' nanokernel work [MK00] achieves sub-microsecond phase errors with respect to an external standard—e.g., a primary frequency reference (Section 2.5) or a radio-based standard such as a GPS or WWVB receiver (Section 2.6). The external references generate "PPS" (pulse-per-second) signals that are timestamped inside the operating system kernel. Strictly speaking, this is not *network* time synchronization—it does not use the (IP) network, and depends on the very low jitter of the external reference hardware. (The NTP software package supports both this mode and normal network synchronization.)

2.4   Clocks and Frequency Standards

So far, we have described schemes that produce various forms of clock agreement among a distributed set of nodes. Implicit to these schemes is the existence of a *clock* local to each node in the system. A clock, in essence, is a device that counts events indicating the passage of some time interval (e.g., 1 second), and uniquely labels each of these intervals (January 10, 1974, 10:14:03 AM). The events are generated by a *frequency standard*, based on the observation of a physical process, such as the transit of the Sun through a local meridian, the vibration of a quartz crystal, the beat of a pendulum, or the resonance of a cesium beam tube.

The quality of a clock usually boils down to its *frequency stability*—that is, the ability of its frequency standard to emit events at a constant frequency over time. The absolute value of the frequency compared to the desired value—or, its *frequency accuracy*—is also important, but calibration can easily compensate for an inaccurate but stable standard.

The focus of our work is on methods of clock *synchronization*, not on con-

struction of better clocks. However, clocks are very important: the error bound achieved by a clock synchronization method is linked to both the error inherent in the method itself, and the stability of the clocks' frequency standards. In fact, to some extent, the two are interchangeable. Stable clocks can compensate for a synchronization channel between them that is prone to large (but unbiased) errors: many synchronization events can be averaged over a long time. Similarly, a precise synchronization channel can compensate for a poor-stability frequency standard; frequent synchronization minimizes the time in-between when the clock is left to fend for itself.

Many types of frequency standards exist. In general, as their stability and accuracy increase, so do their power requirements, size, and cost, all of which are important in sensor networks. Most commonly found in computer clocks are *quartz crystal oscillators*, characterized by Vig in [Vig92]. Quartz crystals are attractive because they are inexpensive, small, use little power, and perform surprisingly well despite their low resource requirements.

The frequency generated by a quartz oscillator is affected by a number of environmental factors: the voltage applied to it, the ambient temperature, acceleration in space (e.g., shock or attitude changes), magnetic fields, and so forth. More subtle effects as the oscillator ages also cause longer-term frequency changes. The inexpensive oscillators commonly found in computers have a nominal frequency accuracy on the order of between 10^4 to 10^6—that is, two similar but uncalibrated oscillators will drift apart between 1 and 100 microseconds every second, or, between about 0.1 and 10 seconds per day [Vig92, EE01]. However, their frequency *stability* is significantly better—with a change in frequency of one part in 10^9 to 10^{11} when averaged over several seconds or more.[1]

---

[1]Historically speaking, this stability is remarkable: a \$5 digital wristwatch can keep time to

There are a number of variations on the basic crystal oscillator (XO). A *temperature-compensated oscillator* (TCXO) contains either an analog circuit or a microprocessor that corrects for temperature variations. An *oven-controlled oscillator* (OCXO) generates heat to keep the system constantly at the particular temperature at which it exhibits the greatest frequency stability. These variations are more stable, but use much more energy, require longer startup times, are larger, and cost many times more (e.g., $100 instead of $1).

Pásztor and Veitch have shown that the crystal oscillators used to drive typical Pentium-class microprocessor cores have inaccurate but fairly stable frequencies, making them useful in high-precision measurement applications [PV02]. The CPU clock drives the Pentium's TSC (instruction cycle) counter, which can be queried with low latency, and has a resolution commensurate with the CPU core's clock frequency—1 nanosecond for a 1 GHz CPU. Because the TSC counter exhibits high frequency stability, the relative frequency of distributed microprocessors can be determined with high precision by averaging over a long timescale. Their method therefore provides a way to accurately measure *relative time intervals*, but does not speak to distributed phase synchronization. (Our RBS scheme benefits from high-resolution and high-stability local oscillators; as we will see in Chapter 6, the standard 1-μsec clock resolution on our iPAQs is likely the limiting factor in RBS's performance. We also hope to use the TSC counter to achieve better results in our future work.)

---

within one second after six months if its wearer first performs an experiment to determine the difference between the watch's frequency and a real second, as seen in [AAH97]. Consider, for comparison, that in the 1714, the British Board of Longitude offered £20,000—the modern-day equivalent of about $5 million—for a method of determining longitude at sea to within half a degree [SA98]. To use a chronometer for this task implied it could gain or lose no more than two minutes over a six-week voyage. The best mechanical clocks of that era typically accumulated 15 minutes of error per day.
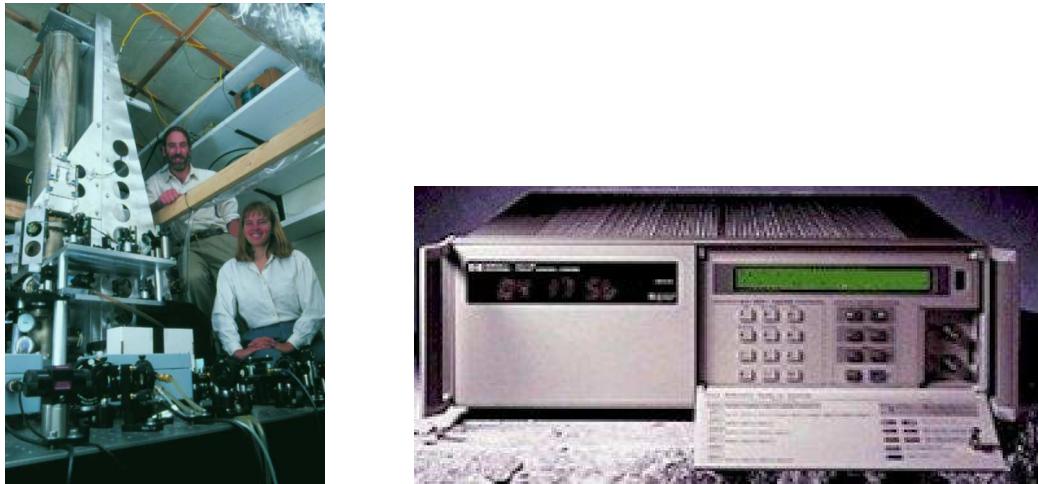
Figure 2.1: *left*) The NIST-F1 cesium fountain atomic clock—the most stable in the world since its completion in 1999, developed by Steve Jefferts and Dawn Meekhof at the U.S. National Institute of Standards and Technology. (Photo copyright 1999 Geoffrey Wheeler, used with permission of NIST.) *right*) A commercially available cesium clock—the Agilent (formerly Hewlett-Packard) 5071A.

2.5 The Definition of a Second

Aside from crystal oscillators, the most important frequency standards for computer clock synchronization are the class of *atomic frequency standards*. Unlike mechanical standards such as pendulums or vibrating quartz crystals, atomic standards are based around the observation of atomic properties of elements such as cesium and rubidium.

Modern implementations of cesium clocks achieve accuracy far beyond that possible with any mechanical clock. The current (2003) state-of-the-art is exemplified by the NIST-F1 cesium fountain created by the United States' National Institute for Standards and Technology. The F1 is a laboratory standard that achieves an error bound of less than 2×10^{-15}, which is about 0.2 nanoseconds per day, or 1 second in nearly 20 million years [NPL02]. This clock is large (Figure 2.1) and custom-built, but less accurate cesium clocks are readily available

commercially. For example, the Hewlett-Packard 5071A costs about $50,000, is the size of an ordinary desktop computer, and is accurate to 1 second in 162,000 years.

The accuracy of these clocks is impressive, but an atomic basis for frequency is attractive for a more fundamental reason. Einstein's equivalence principle predicts that atomic properties are the same in all places and at all times [AG01]. In other words, atomic clocks can act as *primary* standards—independent sources of frequency that are available everywhere, rather than frequency sources that must be calibrated against a prototype. In contrast, for example, the definition of a kilogram is the mass of a particular prototype object stored in France.

This reproducibility (and, hence, accessibility) makes atomic observations an ideal basis for a standard. Thus, in 1967, an atomic definition of time was adopted within the SI system of measurements: *The second is the duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium 133 atom.*

The change to an atomic second was truly a revolution. Throughout most of human history, observation of the heavens was considered to be the final authority on time. Noon was, by definition, when the Sun was directly overhead, and modulo certain details, one second was simply 1/86400 of the average length of a day. Cesium atomic clocks were the first widely available frequency standards that were significantly more stable than the rotation of the Earth itself, and were used to show that the length of a day can vary by 10s of milliseconds [AG01].

Depending on the application, the definition of an SI second may or may not be relevant to computer clock synchronization. There are many situations where distributed nodes need to be *synchronized* but not necessarily running at a particular frequency known *a priori*, as we will see in Section 4.6.

2.6 National Time Standards and Time Transfer

Some applications that need a reference to an SI second use their own primary frequency reference, e.g., a local cesium clock. However, the majority instead refer to the standard definition of a second that has been physically realized elsewhere. In addition, many applications require more than just a reference *frequency* (how long is a second?) but also need to know the *time* (when did 12:00:00 occur?).

In most industrialized countries, government agencies are tasked with maintaining the country's reference frequency, and using that frequency to construct its official timescale. For example, in the United States, the responsibility lies jointly with the U.S. Naval Observatory's Time Service Department and the National Institute of Standards and Technology's Time and Frequency Division. USNO's "Master Clock #2" is steered by an ensemble of over 45 hydrogen maser and cesium frequency standards [MMK99]. UTC, the most commonly used international civil timescale, is based on a weighted average of over 200 atomic clocks at over 50 such national laboratories [Tho97]. It is calculated at BIPM (*Bureau International des Poids et Mesures*) in Paris, France.

Many of these laboratories are involved in active research in the area of *time transfer*—that is, wide-area dissemination of both frequency and time. This capability is a fundamental requirement for any national laboratory whose mission is to transmit time and frequency to outside users (industry, the military, private individuals, and so forth). It is also the technology that enables computation of aggregate time standards such as UTC. Many modern time transfer schemes have been developed.

Historically speaking, they all harken back to much older methods. An unprecedented surge of interest in timekeeping was seen in the 14th century as

mechanical clocks swept Europe, often found in the belfries and towers of city halls and churches [Lan83]. Clocks in other forms had been known for at least a thousand years beforehand, such as the sundial or water-driven *clepsydra*. But these newer mechanical clocks were the first to be accompanied by automatically ringing bells—a simple form of time transfer. In 1845, the U.S. Naval Observatory began to disseminate time by dropping a "time ball" from atop a flagpole every day precisely at noon [BD82]. The ball was often used by ships anchored in the Potomac river that needed a calibrated chronometer for navigation before setting out to sea [SA98].

In modern times, the state of the art is in radio time transfer methods. The first of these, seen originally in the 1920's, was a voice announcer on radio station WWV operated by the National Institute of Standards and Technology. An automated version of WWV is still in operation today, along with its more recent cousin, WWVB, which provides computer-readable signals in stead of voice announcements. The modern-day service provides references to U.S. time and frequency standards within one part in 10^{13} [Bee81].

Satellite navigation systems have an important relationship to time—most notably, the United States' NAVSTAR Global Positioning System, commonly called GPS [Kap96]. GPS provides localization service by allowing users to measure time of flight of radio signals from satellites (at known locations) to a receiver. As we will discuss in the next section, the result of the localization computation includes the offset between the user's clock and GPS system time as a "side effect."

Since GPS system time is kept within several nanoseconds of the UTC(USNO) timescale, GPS has emerged as the pre-eminent method for distributing UTC time and frequency world-wide. Dramatic technological progress in receiver hard-
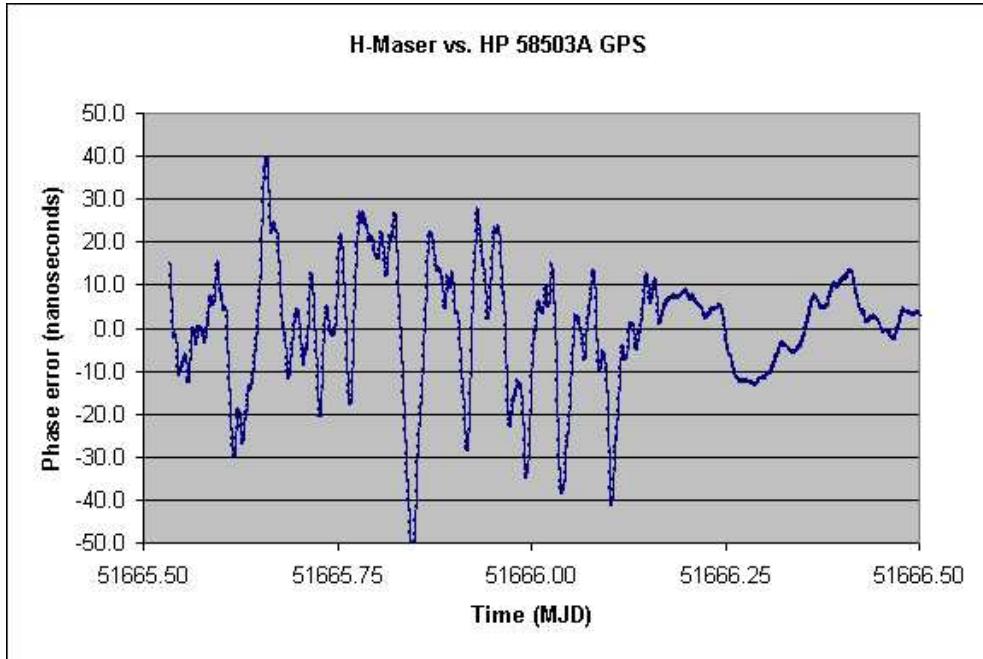
Figure 2.2: The phase error between the PPS outputs of a Hewlett-Packard 58503A GPS-steered clock and a Kvarz CH1-76 Passive Hydrogen Maser on May 1, 2000. (A hydrogen maser exhibits exceptional short-term frequency stability, but its long-term frequency must be tuned to a primary cesium standard.) The removal of Selective Availability near MJD 51666.25 can be seen to dramatically reduce the jitter in the GPS clock's output. *Figure courtesy of Tom Van Baak at LeapSecond.com, used with permission.*

ware has brought about both significant improvments in timing accuracy and a precipitous drop in cost. By 1999, a typical receiver could achieve 200nsec accuracy with respect to UTC [MKM99]. Further, on May 1, 2000, U.S. President Bill Clinton announced the removal of Selective Availability (SA) from GPS service (Figure 2.2). SA is an intentional degradation of the GPS signal designed to prevent civilians from accessing military-grade time and position services. In 2003, without SA, a relatively inexpensive ($500) GPS clock can achieve < 20nsec phase error to UTC [CNS].

The most active subject of modern-day research in time transfer is *two-way*

satellite time transfer [SPK00]. TWSTT allows two clocks connected by a satellite relay to both compute their offset from their peer. It is superior to methods such as GPS time or WWV's radio broadcasts because it transfer is *two-way*, instead of solely from sender to receiver. TWSTT methods are used to connect the time and frequency standards of national laboratories to BIPM in France for computation of international timescales.

One example of TWSTT is "GPS Common View," which can define the relationship of two user clocks to *each other* with greater accuracy than GPS can provide UTC [LL99]. Using the orbital parameters of the satellites in the GPS constellation, two sites that wish to synchronize pick a time when a particular satellite will be within line-of-sight to both locations. The two sites then compare the phase of the observed satellite broadcast to their own local clocks, correcting for time-of-flight by computing the difference between the clock's position and the satellite's position. (The satellite's position is computed throughout the course of the experiment using the orbital parameters contained within all GPS satellite broadcasts.) This technique does not deliver UTC to either clock, but does define the phase and frequency *difference* between the two clocks, which is the goal. The precision is high because the two sites have a "Common View"—that is, they are observing the same physical signal that was emitted at the same time. We will see in Chapter 6 that schemes such as CesiumSpray and our single-hop version of RBS resemble this technique: accuracy is improved by removing uncertainty at the sender from the critical path.

2.7   Methods for Avoiding the Problem

In some cases, the time synchronization problem can be solved by *not* solving it—designing a system to avoid the need for an explicitly synchronized clock. We will

illustrate this philosophy by describing two types of systems that exemplify it: Lamport's idea of virtual clocks, and three different spatial localization systems that use different tricks to work without synchronized clocks.

2.7.1 Virtual Time

A landmark paper in computer clock synchronization is Lamport's work that elucidates the importance of *virtual clocks* in systems where causality is more important than absolute time [Lam78]. In his system, each computer has a local clock which is incremented monotonically after each event that it observes. Each message sent to another node also carries the timestamp of the sender with it. When a node receives a message, it advances its local clock to a value greater than the value in the received message. This simple scheme is enough to guarantee that the timestamps on messages can be used to reconstruct the total ordering of any sequence of events that were causal. That is, if event A causes event B, the timestamp of A is less than the timestamp of B.

Lamport's work focused on giving events a total order rather than quantifying the time difference between them. While it is often not enough for sensor network applications that require *physical* time synchronization, it has emerged as an important influence. Many sensor applications require only *relative* time. For example, when timing the propagation delay of sound to determine a range [GE01], reference to an external time standard such as UTC may not be relevant.

2.7.2 Localization Without Synchronized Time

Time and space are closely intertwined; spatial localization systems are often intimately tied to time synchronization. Many systems measure ranges by measuring the time-of-flight of some phenomenon: the system emits a recognizable

signal at a sender S, then times the interval required for the signal to arrive at a receiver R. If the propagation speed of the phenomenon is known, the time measurement can be converted to distance.

In such systems, there is often a circular dependency between the method for performing time synchronization, and the need for synchronized clocks to perform the time-of-flight measurement. It is instructive to consider three such systems that use different tricks for breaking this dependency:

- PinPoint implicitly provides time synchronization between S and R because they are both the same device.

- GPS avoids the need for time synchronization between by S and R by over-constraining other relationships between the two.

- Active Bat explicitly time-synchronizes S and R using a modality whose propagation speed is much faster than the ranging signal.

We will now briefly describe each of these systems in a bit more detail.

PinPoint RF-ID System

PinPoint, Inc.'s RF-ID system [WL98] avoids the synchronization problem by setting $S = R$, i.e., making S and R the same node. The RF-ID system consists of two types of components: a *base* that acts as both a signal emitter and receiver, and a *tag* one that acts as a reflector. The bases are permanently mounted in the environment, in known locations within a 3D coordinate system.

The bases periodically send interrogation signals. The tags can reflect the signals back to the bases with extremely low and deterministic delay, modulated with data that identifies the tag. Bases can estimate the one-way time-of-flight to

the tag as half the round-trip-time for the reflected signal. A range from the base to the tag can be computed based on the time-of-flight and c, the RF propagation speed. After measuring several ranges from different bases at known locations, the tag can be located in 3-space by multi-laterating.

GPS

The GPS satellite system, mentioned in Section 2.6, also localizes by measuring the time-of-flight of RF signals [Kap96]. In this case, S is the GPS system (the satellite constellation) and R is the user's receiver (e.g., a hand-held device).

Each GPS satellite is in a known position relative to an Earth-based coordinate system. The orbits are monitored by ground-based tracking systems; orbital parameters are published in real-time using the GPS signal itself. As with Pin-Point, the goal is to locate a user in 3-space by measuring the range from each of these known positions to the user, and multi-laterating.

If some (imaginary) form of time synchronization between the satellite constellation (S) and the user (R) existed, it would be easy to measure the ranges between them. A satellite could send a signal, and tell the user when it was sent according to the (imaginary) global clock. The user's range to the satellite could be computed based on the interval between the emission of the signal and the time the user received it. Ranges from three satellites to the user would be needed to solve for a three-dimensional (X, Y, Z) solution.[2]

Since no such synchronization exists, GPS instead requires *four* satellites, and solves for a *four*-dimensional solution (X, Y, Z, T), where T is the *unknown* offset

---

[2]Three spheres actually intersect at two points, not one. But, one of these points is near the surface of the Earth, while the other is in space. The incorrect answer can be easily recognized and discarded.

between GPS system time and the user's clock. Therefore, by over-constraining the relationship between S and R, GPS prevents the need for S and R to share a synchronized clock—indeed, such synchronization is provided as an output. However, it is important to note that time synchronization is not completely avoided: for the scheme to work, the GPS satellites (i.e., members of S) need to be synchronized with *each other*. Without this inter-satellite synchronization, each satellite used by the receiver would add both an equation *and* an unknown to the system. By keeping the satellites in sync, there is only a single unknown T, regardless of how many satellite observations are used by the receiver.

Active Bat

Finally, ORL's Active Bat system [WJH97] demonstrates a third way of breaking the circular dependency between localization and time. The Active Bat uses a time-synchronization modality between S and R that is different than the time-of-flight measurement modality. Modulo certain details, S simultaneously emits a (fast) RF pulse and a (slow) ultrasound pulse. The time of flight of the RF pulse is negligible compared to that of the ultrasound pulse. The RF pulse can be considered an instantaneous phenomenon that establishes synchronization between S and R with respect to the ultrasound signal.

While this solution seems simple and attractive, we will argue in Chapter 9 that keeping explicitly synchronized clocks actually can reduce the overall system complexity in this case. Our acoustic ranging system uses RBS to keep clocks synchronized. Having an explicit and self-contained synchronization service that is separate from the acoustic subsystem has a number of benefits. For example, the system becomes more decomposable and modular because there is no longer a tight interdependence between the acoustic process and the radio.

2.8 Summary

In this chapter, we have reviewed a broad spectrum of work in synchronization of computer clocks in both their time (phase) and frequency. Naturally, this begs the question: *why do we need something new?* The essence of the answer lies in the fact that many of these methods rely on assumptions that are violated in the new sensor network regime. However, before examining this claim in more detail, we must first answer one other prerequisite question: what uses are there for synchronized time in a sensor network?

CHAPTER 3

The Need for Synchronized Time

Come what come may,

Time and the hour run through the roughest day.

—William Shakespeare

Why do sensor networks need synchronized time?

Before discussing time synchronization *methods*, we must first clarify its *motivations*. In this chapter, we will describe some common uses of synchronized time in sensor networks. The wide net cast by the disparate application requirements is important; we will argue in the next chapter that this diversity precludes the use of any *single* synchronization method for all applications.

We begin in Section 3.1 by formulating the need for time synchronization in sensor networks in its most generic terms: synchronized physical time is a fundamental requirement if our goal is to reason about a time-varying environment that is observed by distributed sensors. Most emerging applications are special cases of this single rule. In Section 3.2, we show that energy constraints make in-network synchronization truly unavoidable because data reduction must happen inside the network.

In Section 3.3, we describe the converse situation: coordinated actuation. While sensor integration *observes* the state of the world, actuation *changes* it.

Synchronized time is vital for coordinating actions across a distributed set of actuators.

Sections 3.4, 3.5, and 3.6 describe some specific applications of our general rules: energy-efficient radio scheduling, acoustic ranging, and distributed signal processing.

Finally, in Section 3.7, we review some of the traditional uses for synchronized clocks in distributed systems. At its core, a sensor network is also a distributed system, and many of the same problems appear in both domains.

3.1 Multi-Sensor Data Integration

In Chapter 1, we described the situations that most often motivate sensor network research: phenomena of interest that span a large geographic region, but can not be sensed from far away. Remote sensing might be impossible because the environment is cluttered, leaving only short line-of-sight paths, or because the signal of interest simply can't propagate very far through the environment. In these situations, *distributed* sensing must be accomplished through placement of large numbers of sensors close to the phenomena of interest.

One of the most powerful aspects of the design of such a network is *multi-sensor integration*—the combination of information gleaned from multiple sensors into a larger world-view not detectable by any single sensor alone. For example, consider a sensor network whose goal is to detect a stationary phenomenon P, such as that depicted in Figure 3.1. P might be a region of the water table that has been polluted, within a field of chemical sensors. Each individual sensor might be very simple, capable only of measuring chemical concentration and thereby detecting whether or not it is within P. However, by integrating knowledge
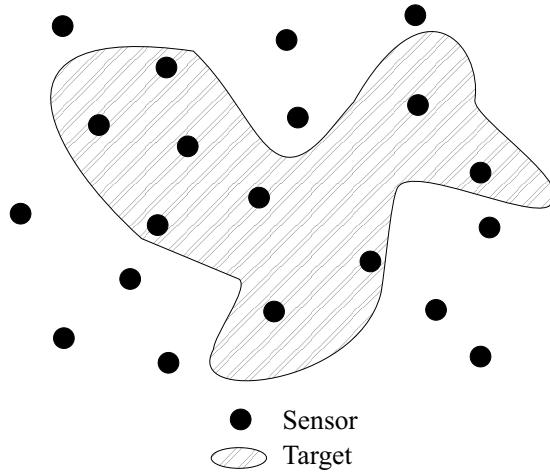
Figure 3.1: Individually, sensors may only be capable of a binary decision: they are within the sensed phenomenon, or they are not. If the sensor positions are known, integration of information from the entire field allows the network to deduce the *size* and *shape* of the target, even though it has no "size" or "shape" sensors.

across all the sensors, combined with knowledge about the sensors' positions, the complete network can describe more than just a set of locations covered by P: it can also compute P's *size*. The whole of information has become greater than the sum of the parts: the network can deduce the size and shape of P even though it does not have a "size" or "shape" sensor.

3.1.1 The brute-force approach

This type of collective behavior does not always require synchronized time. This was the case in the example above—because P was not moving, the situation was time-invariant. However, what if P is mobile, and the objective is to report P's speed and direction? At first glance, it might seem that the localization system we described above can be easily converted into a motion tracking system by performing repeated localization queries over time. If the data indicating the location of P all arrive at a single, central location for integration, we might

conclude that no time-synchronization of the sensors themselves is required. The central node can simply (locally) timestamp each localization reading as it arrives from the field. The timestamped locations can then be integrated into a track that indicates speed and direction.

Such a brute force approach can work—in some situations. One example is an asset tracking system that can locate pieces of equipment within an office building. If we define the "motion" of an object as the history of all the rooms in which it has been located, an equipment motion tracker might be implemented by asking the object tracker for the equipment's location several times a day, and compiling a list of offices in which the equipment was spotted over time. This works if we assume that the travel time of messages from the equipment sensors back to the integration point was zero. In other words, if the system is asked, "Where is object X?" we must assume it can generate a reply that is, in effect, instantaneous and still correct when it is received. Such an assumption is probably valid for asset tracking: objects are likely to stay in the same place for hours or days—extremely slowly compared to the time needed by the object-tracker to locate an object and report its location to the user or a data integrator.

3.1.2   Failure of the brute-force approach

Unfortunately, as the message latency grows relative to the speed of the phenomenon being tracked, the brute-force approach fails. Consider a network in which the range of possible message latencies (i.e., the jitter) is denoted by time interval j. Using the brute-force approach, it is impossible to track a mobile target with more precision than the distance that it can cover during time j. This can be a serious limitation in sensor networks: j can be very large, and phenomena can move quickly.

j is large because sensors are distributed over a wide area, but their communication range is very short relative to the size of that area. Information can only travel through such a network hop-by-hop, so the latency from a sensor to a central integrator will vary with the number of hops that separate them. The small communication range is a fundamental limitation, driven primarily by the need for energy conservation: long-distance transmission has a disproportionate energy cost (r^2 to r^4). Energy constraints have other effects on latency: sensor networks use low-power radios that often have very low bit-rates, thus magnifying the delay caused by multi-hop routing. Wireless medium-access protocols often trade latency for energy savings, through duty-cycling and other techniques. Significant delay can be introduced by retransmissions as well, as packet loss is common in wireless networks. Links may even become unusable over long time intervals.

In addition to large values of j, the other side of the equation is also a problem: fast-moving phenomena are a reality of nature. For example, in applications that require acoustic localization or beam-forming (Chapter 9), there is no avoiding the speed of sound. Even during a small j interval, sound can travel a distance that may be larger than the desired spatial accuracy of the system.

These limitations suggest that sensor observations must be time-stamped *inside of the network*, as near as possible to the original event. Doing so dramatically reduces the inaccuracy of timestamps because the many sources of jitter in message delivery are no longer in the critical path. However, in-network timestamping implies that, unlike in the brute force solution, the sensors themselves must be time-synchronized.

3.2 In-Network Processing

In the previous section, we argued that the integration of observations from sensors throughout the network requires time synchronization of the sensors themselves. Jitter makes synchronization necessary even in a simple centralized-processing scenario. However, such a scenario is unrealistic, and the role of time synchronization becomes even more important in realistic designs.

Centralized processing is infeasible in sensor networks; such a design assumes there is a practical way to transmit every raw sensor value generated throughout the network to a single node that is responsible for processing it. Even in traditional distributed systems, this would be unwise: scaling tends to be clumsy or impossible, and a single point of failure makes the system much less robust. In sensor networks, centralization is particularly fatal due to a requirement not seen in other distributed systems: energy efficiency.

We saw in Chapter 1 that energy efficiency is crucial because it defines the longevity of the system. As communication is the primary consumer of energy [PK00], and the low power budget of the nodes precludes communication beyond a short distance, much research has been focused on maximizing local processing and minimizing the overhead of collaboration. Emerging designs allow users to task the network with a high-level query such as "notify me when a large region experiences a temperature over 100 degrees" or "report the location where the following bird call is heard" [HSI01, MFH02].

Such tasking can dramatically reduce the communications overhead (and, thus, the energy use) by allowing sensor nodes to perform expensive computations locally. For example, nodes can correlate an incoming audio stream to the desired pattern, and report only the time and location of a match. This is many orders of

magnitude more efficient than transmitting the complete time-series of sampled audio. Unnecessary communication back to the user may be further reduced by forming collaborations among spatially local nodes so that a match detected by only one of them can be recognized as spurious.

The use of local processing, hierarchical collaboration, and domain knowledge to convert data into increasingly distilled and high-level representations—or, *data reduction*—is key to the energy efficiency of the system. In general, a perfect system will reduce as much data as possible as early as possible, rather than incur the energy expense of transmitting raw sensor values further along the path to the user.

There are many examples of how different data reduction processes depend on time synchronization. One is *duplicate suppression*, or, collaboration that prevents redundant notification of an event by more than one member of a group of nearby sensors that observed it. Intanagonwiwat *et al.* showed this technique can produce significant energy savings [IGE00]. Synchronized clocks are needed to resolve to otherwise ambiguous cases, as shown in Figure 3.2: Did the sensors see different views of a single event? Or did they see two distinct events?

In many cases, in-network data reduction takes the form of multi-sensor integration, as we described in Section 3.1. For example, where possible, the group of sensors that detect a tracked phenomenon should also compute its speed and direction. Transmission of a locally computed velocity vector is far more efficient than sending a complete time-series of proximity detections from every sensor to a far-away processing point. The improvement is more significant if the target area is many hops away from the ultimate destination of the data (e.g., the user). For such data reduction to be possible, the sensors near the target must be time-synchronized.
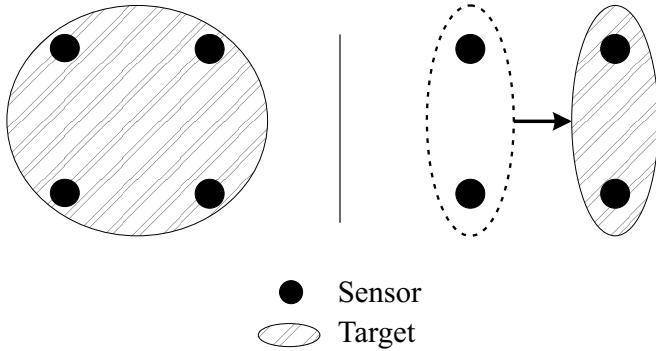
Figure 3.2: A network tracking a target may wish to eliminate redundant sightings when the target is in view of more than one sensor. *left*) All four sensors see a single, stationary target: only one event should be reported. *right*) A mobile target is sighted by two sensor pairs at different times: two distinct events should be reported. To disambiguate these two cases, the sensors must compare their detection times. This implies the need for synchronized clocks.

3.3   Coordinated Actuation

Most of our discussion of sensor networks speaks of them as entirely passive systems—capable of detecting the state of the environment, but unable to change it or the network's relationship to it. However, actuation can dramatically extend the capabilities of a network, and a number of research groups have made significant progress in that direction.

One of the most commonly described forms of actuation is sensor mobility. Sibley *et al.* have built a prototype RoboMote, which brings autonomous mobility to the Berkeley Mote sensor platform [SRS02]. They also describe its use in *connectivity-based robotic sensor networking*—using changes in connectivity between nodes to understand their relative positions. Rahimi *et al.* analyze the feasibility of using mobile robots to adaptively harvest energy from the environment [RSS03]. Sukhatme *et al.* propose using mobile sensors for *actuated boundary detection*—the extent of a target can be more precisely determined if the sensors are mobile, as shown in Figure 3.3.
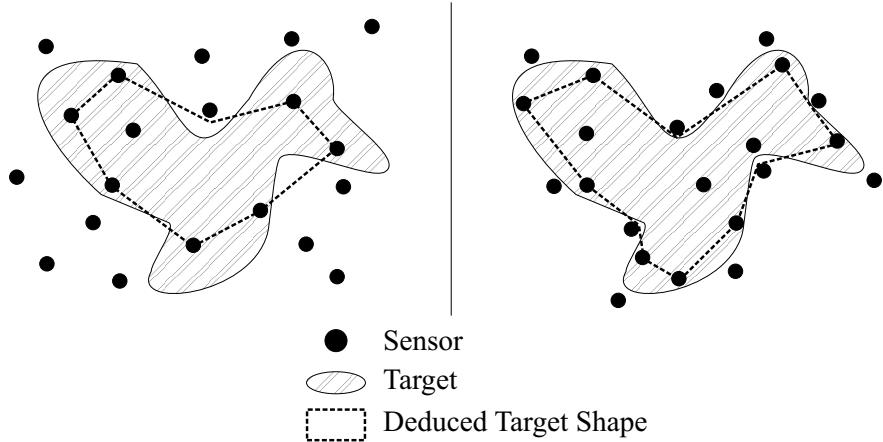
Figure 3.3: *left*) A sensor field might deduce the shape of a target area (e.g., contaminated soil) by drawing a convex hull around the sensors that are within the contamination, with concave deviations around uncontaminated sensors. *right*) If the sensors are *mobile*, they can collaboratively adjust their positions, estimating the shape of the target with greater accuracy.

Mobility is also central to the DARPA SHM program [SHM]. Nodes collaboratively draw a map of their relative positions via acoustic localization. Each node is equipped with a mobility subsystem (small rocket motors); if a gap in the sensor field is found, nearby nodes move in to fill the space. This system will be described in more detail in Section 9.2.

Of course, there are many types of actuation other than mobility. For example, sensors may emit sounds, release chemicals, apply pressure to support structures [BCY98], or simply turn on and off [CE02]. In fact, the energy-efficient radio scheduling we will describe in Section 3.4 is actually a special case of coordinated actuation.

Broadly speaking, systems that incorporate actuation require synchronized time in two ways. First, a sensor's decision to act is motivated by knowledge from the same sorts of in-network data reduction and multi-sensor integration algorithms that are used by static sensor networks. These algorithms require

synchronized time, as we have already described. The second use comes from *coordinated* actuation; if more than one node plans to act, the actions often need to be coordinated with each other. As with sensing, coordination of events in the physical world requires a common reference frame in physical time and space.

3.4 Energy-efficient Radio Scheduling

Low-power, short-range radios of the variety typically used for wireless sensor networks expend virtually as much energy passively listening to the channel as they do during transmission [PK00, APK00]. As a result, MAC (Medium ACcess) protocols are often designed around this assumption, aiming to keep the radio off for as long as possible. TDMA is a common starting point because of the natural mechanism it provides for adjusting the radio's duty cycle, trading energy expenditure for other factors such as channel utilization and message latency [Soh00]. Precious energy can be conserved by turning the radio off, waking up only briefly to exchange short messages before going back to sleep.

The energy savings is often directly correlated with the precision of time synchronization. Consider two wireless sensor nodes that have agreed to rendezvous on the radio channel once every 60 seconds to exchange a short message—say, 8 bits representing the current temperature. Using a 19.2kbit/sec radio such as our testbed's RF Monolithics [CEE01], 8 bits can be transmitted in about 0.5ms. However, in practice, the radio must be awakened early to account for time synchronization error, as depicted in Figure 3.4.

An expectation of only 1ms phase error will triple the total amount of time the radio is expending energy listening to the channel. The factor is even larger for faster radios—with a 56kbit/sec radio, a 1ms guard band is more than 7
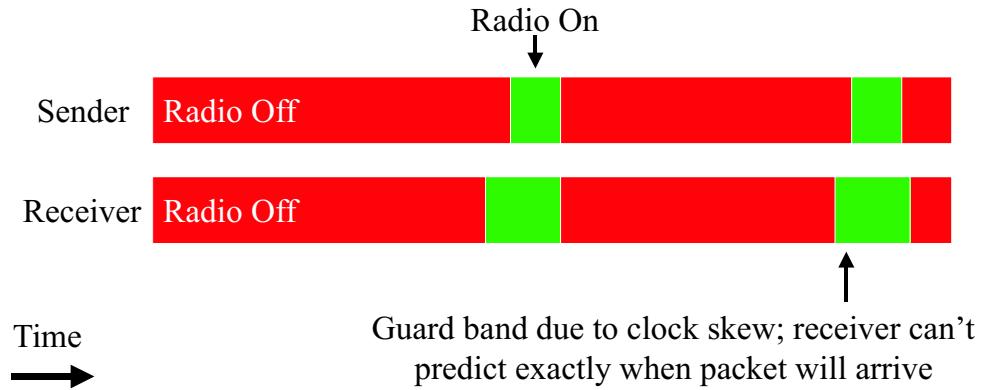
Figure 3.4: The effect of time synchronization on the efficiency of a TDMA radio schedule. When nodes pick a rendezvous time for data exchange, a "guard band" is required to compensate for imprecise clock synchronization. In sensor networks, the guard band can be a major source of energy expenditure: applications may transmit very few data bits, separated by long sleep times.

times longer than the time needed to transmit 8 bits. In addition, even assuming perfect synchronization at the *start* of a sleep period, a typical quartz oscillator on such a sensor will drift on the order of 1 part in 10^5 [Vig92], or 0.6ms after 60 seconds. Of course, sending synchronization packets during the sleep period defeats the purpose of sleeping, so we must consider frequency estimation as part of the time synchronization problem.

This example demonstrates not only the importance of time synchronization in a sensor network, but also one of its difficulties: any resource expended for synchronization reduces the resources available to perform the network's fundamental task. Traditional TDMA systems (e.g., cellular telephone networks) often do not have this constraint, and are engineered only to maximize channel utilization. Good time synchronization is important in those systems because it reduces the size of the guard time, but it is also easier because of the high data rate: each frame received also implicitly gives information about the sender's clock. This information can be used to frequently re-synchronize a node with its peers [LS96].

3.5 Acoustic Ranging

Spatial localization is a crucial building block for many sensor network applications. Indeed, virtually all of the application examples in this chapter—multi-sensor integration, array processing, coordinated actuation, and so forth—require a common reference frame in both time *and* space. In the smallest networks, it is possible to implement spatial localization of nodes manually—by carefully measuring and "hard-coding" the locations of the sensors. However, this is approach does not scale, is ill-suited to ad-hoc deployment, and does not work in networks with mobile nodes. Consequently, many researchers have focused attention on building systems that discover location autonomously [WHF92, BHE03, GBE02, WC02].

One common localization method is based on *acoustic ranging*, where a node emits a sound that can be recognized by other nodes in the area. If the nodes are time-synchronized, they can compute the time elapsed between the emission and reception of the sound. Using an estimate of the speed of sound through the medium, this delay can be converted into a distance. A maximum-likelihood coordinate system can be constructed based on the constraints provided by many such pairwise distance measurements. We have implemented such a system, which is described in detail in Section 9.1.

3.6 Array Processing

For decades, the signal processing community has devoted enormous research attention to seamless integration of signals from multiple sources, and sources with heterogeneous sensing modalities. The signal processing literature sometimes refers to this as *array processing*; with heterogeneous sensors, it is often
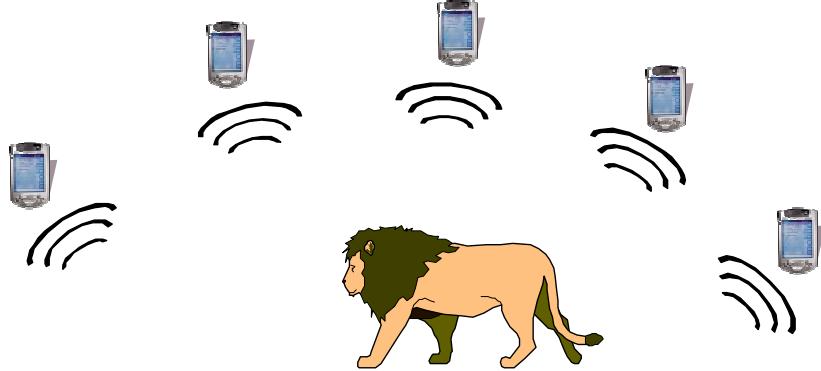
Figure 3.5: Acoustic beam-forming arrays localize the source of sound generated by a source that is not time-synchronized with the array. Similar to the trick used by GPS (Section 2.7.2), the system is over-constrained if the microphones are in known positions and are synchronized with each other. Localization is possible solely based on time-of-arrival *differences* across the array.

called *data fusion*. There are many applications, such as signal enhancement (noise reduction), source localization, process control, and source coding. It would seem to be a natural match to implement such algorithms in distributed sensor networks, and there has been great interest in doing so. However, much of the extensive prior art in the field assumes *centralized sensor fusion*. That is, even if the sensors gathering data are distributed, they are often assumed to be wired into a single processor. Centralized processing makes use of *implicit* time synchronization—sensor channels sampled by the same processor also share a common timebase.

For example, consider the "blind beamforming" array for localizing the source of sound, described by Yao *et al.* [YHR98]. Their array computes phase differences of acoustic signals received by sensors at different locations. From these phase differences, it can compute differences in the time of flight of the sound from the source to each sensor (Figure 3.5). This allows the sound's source to be localized with respect to the spatial reference frame defined by the sensors

in the array. However, their technique makes the implicit assumption that the sensors themselves are time-synchronized with each other. In other words, the beam-forming computation assumes that the observed phase differences in the signal are solely due to differences in the time of flight from the sound source to the sensor. The technique breaks down if there are variable delays incurred in transmitting signals from the sensor to the processor. In a centralized system, where there is tight coupling from sensors to a single central processor, it is usually safe to assume that the different sensor channels are in synchrony. For such an array to be implemented on a fully distributed set of autonomous wireless sensors, *explicit* time synchronization of the sensors is needed.

It is interesting to note that a blind beam-forming array makes use of a trick similar to GPS, as we described in Section 2.7.2. In the acoustic ranging we described in Section 3.5, the acoustic emitter and receiver were collaborative. In contrast, a lion can not be time-synchronized with an acoustic array, just as the GPS satellite constellation is not synchronized with GPS receivers. The acoustic array works by over-constraining the system; similar to GPS, the time at which the sound was generated is simply an unknown in the system of equations.

In Section 9.3, we will show how our time synchronization methods were a key factor that enabled the implementation of the first fully-distributed version of the centralized array described here.

3.7 Traditional Uses in Distributed Systems

The uses of time synchronization we have described so far have been specific to sensor networks, relating to their unique requirements in distributed signal processing, energy efficiency, and localized computation. However, at its core, a

sensor network is also a distributed system, where time synchronization of various forms has been used extensively for some time. Many of these more traditional uses apply in sensor networks as well. For example:

- *Logging and Debugging.* During design and debugging, it is often necessary correlate logs of many different nodes' activities to understand the global system's behavior. Logs without synchronized time often make it difficult or impossible to determine causality, or reconstruct an exact sequence of events.

- *State Consistency.* Some systems use distributed agreement or voting algorithms—polling a set of servers to find the most recent state, despite node or network failures that might prevent state updates from reaching every server [Gif79]. Synchronized time can play a key role in these algorithms if state updates are emitted by more than one node.

- *Database Queries.* A number of research groups have extended traditional database query semantics to the sensor network domain [BGS00, MF02]. These schemes seek to hide the complexity of the network from the user by supporting distributed queries as if from a centralized database. To maintain the same query semantics across a distributed set of nodes implies the need for synchronized time, as Madden and Franklin described [MF02].

- *Interaction with Users.* People use "civil time"—their wristwatches—when making requests such as "show me activity in the southeast quadrant between midnight and 4 A.M." For this request to be meaningful, at least a subset of the nodes in a sensor network need to be synchronized with a civil time standard such as UTC.

- *Cryptography.* Perhaps due to sensor networks' applicability to military missions, there has already been significant interest in applying cryptographic schemes to them [PSW01, EG02, LEH03]. Authentication schemes often depend on synchronized time to ensure freshness, preventing replay attacks and other forms of circumvention. However, Davis and Geer point out in [DGT96] that, in some contexts, a challenge-response protocol can take the place of accurate timestamps.

3.8 Summary

In this chapter, we described how synchronized time is a critical service in sensor networks—it is a basic requirement for virtually all algorithms that seek to reason about time-varying observations made by distributed sensors. Target tracking, energy-efficient radio scheduling, beam-forming, acoustic localization, and coordinated actuation all need synchronized time in various forms. As sensor networks are distributed systems, the traditional requirements are also seen: synchronized clocks are assumed in areas of debugging and system analysis, interaction with users, consistent state update protocols, and cryptography.

It seems clear that *some* form of time synchronization is needed in sensor networks. However, the applications we described are quite varied. To better understand how to satisfy their needs, we must first try to quantify their requirements. That is, we must describe a standard set of metrics that can be applied both to applications that need synchronization, and the services that provide it.

CHAPTER 4

Dimensions of Sensor Network Time

A man with a watch knows what time it is. A man with two watches is never sure.

—Segal's Law

In the previous chapter, we saw a wide range of sensor network applications—acoustic ranging, integration of position estimates into a velocity, in-network duplicate suppression, energy-efficient radio scheduling, and so forth. Each application places different demands on a time synchronization service. Network implementations also vary significantly: unlike the Internet, which presents a single service interface to all users, sensor networks are highly application-specific. Each can be unique in its task details, available hardware and resource constraints, scale of deployment, and expected lifetime.

This variation makes it impossible to characterize the exact requirements for synchronized time in sensor networks in general. However, a number of common metrics arise that we have found useful—both for decomposing the synchronization required by an application, and that which is provided by a particular synchronization method. In this chapter, we will explore these metrics in detail.

4.1 Phase Error

Phase error is the simplest and most commonly used synchronization metric: if the goal is to keep clocks set to the same value, how far apart are they in reality?

There are a number of variations. *Pairwise* error is the time difference between two nodes' clocks. *Group dispersion* is the worst of the $\binom{n}{2}$ pairwise errors in a group. As we will discuss in Section 4.6, error may also be with respect to an external standard.

Applications are sometimes more sensitive to the *worst* case than the average case. For this reason, studies in the literature often report both the average and various percentile bounds; for example, the worst error seen in 95% of trials.

As we discussed in Sections 3.1 and 3.2, the tolerable phase error in an application is typically related to the rate at which the sensed phenomena in the environment are changing. For example, in a motion tracking application, error tolerance is informed by the speed of the target relative to the sensors' effective range and the desired spatial precision of the result. To suppress redundant notifications of the same event by multiple sensors, the error tolerance is driven by the event frequency; this might be hours or milliseconds, depending on the application.

Of applications currently common in sensor networks, the most stringent error bound probably comes from acoustic ranging. Typical audio codecs sample the channel at 48KHz, or 20.8 μsec per sample. At room temperature, sound travels about 7.1mm in this interval. As sub-centimeter spatial accuracy is often the goal [GE01], ranging applications demand the phase error be small enough to correlate individual audio samples with each other—or, a maximum error of about 10μsec. Seismology applications measure the propagation of seismic wavefronts, which

also propagate quickly—about ten times faster than sound. However, the spatial precision requirements in that domain are much lower. Phase errors on the order of a millisecond are acceptable, implying about 3m of spatial error.

Traditionally, GPS receivers have been used in situations where microsecond-scale phase accuracy is required. As long as Selective Availability remains off, modern GPS receivers can typically reproduce UTC within about 20 nanoseconds (0.2 μsec), as we described in Section 2.6.

4.2 Frequency Error

Phase error is an instantaneous metric: at a given moment, how far apart are these clocks? *Frequency* error, in contrast, measures the difference in the clocks' *rates*. This metric is important because it predicts the growth of phase error over time. That is, if clocks are perfectly synchronized *now*, at what rate are they drifting apart? How well will they be synchronized in 60 seconds?

As we discussed in Section 2.4, all clocks have an internal frequency standard that defines their "natural" rate. Some time synchronization methods explicitly estimate the difference between the clock's native and desired frequencies; others do not. Either way, most disciplined clocks[1] look like the example depicted in Figure 4.1, the main difference being the value of τ. For example, a trivial (and surprisingly common) method for synchronizing a clock to a GPS receiver is to set $\tau = 1$ second. That is, every time the receiver generates a PPS (pulse-per-second) signal, the clock is abruptly set to the correct value. In contrast, NTP estimates the difference between the local and target frequencies so that it can apply a series of small corrections on a much shorter timescale (e.g., on every clock tick,

---

[1]One exception is a computer with a VCXO, or Voltage Controlled Oscillator. Such hardware allows the oscillator's frequency output to be tuned by varying the voltage of one of its inputs.
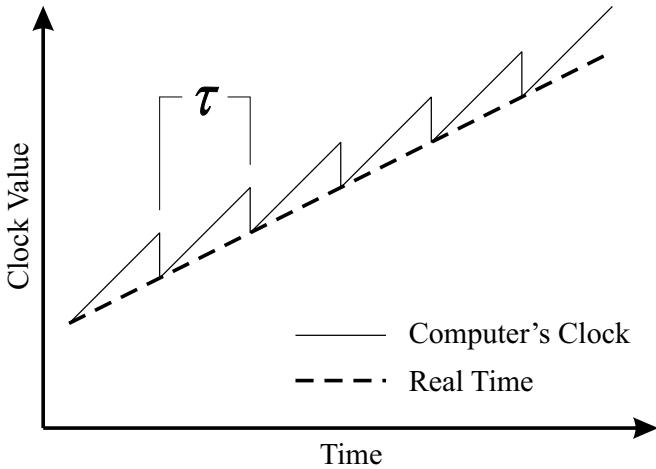
Figure 4.1: A computer clock disciplined to a rate other than its native frequency. Some algorithms simply jump the clock into phase after relatively long intervals (e.g., $\tau = 1$ second). In contrast, algorithms such as NTP explicitly estimate the difference between the native and desired frequencies so that τ can be kept short (e.g., 50 msec). This reduces timescale discontinuities visible to the application, and degrades more gracefully if contact with the frequency reference is lost.

such that $\tau = 50$ milliseconds). NTP can discipline a node's clock based on the PPS output of a local reference such as a GPS receiver, or information received via the network from other nodes running NTP.

In general, algorithms that estimate frequency are superior to those that do not. Keeping τ short makes it far less likely that applications will see discontinuities in the timescale, or grossly incorrect frequencies over short intervals. In addition, if the reference clock becomes unavailable, algorithms that estimate frequency will experience a much slower accumulation of phase error than algorithms that work solely by periodic phase corrections. This is a particularly important factor for sensor networks, where connectivity is often intermittent.

4.3 Lifetime

Lifetime is the interval that clocks remain synchronized, or the interval over which a particular timescale is defined. Sensor networks need synchronization over a wide range of lifetimes, ranging from less than a second to timescales that are persistent.

For example, recall the blind beam-forming array depicted in Figure 3.5. The array localizes the lion by measuring the time of flight of sound—e.g., the lion's footsteps. In this case, sensors within "earshot" of the lion need only be synchronized *momentarily*. In other words, the required lifetime of the synchronization is only the fraction of a second required for the sound to reach all the sensors from its source.

In contrast, consider an extended array that tracks the lion over long time as it moves through the environment (Figure 4.2). To compute the lion's speed or direction over longer timescales, the synchronization lifetime must also be longer—minutes or hours. Luckily, the lion itself moves much more slowly than the sound it makes, making the phase error tolerance of this longer-lifetime synchronization far more relaxed.

Other applications require lifetimes that are still longer: coordinated actuation and tasking by users may require synchronization that lasts the lifetime of the network. In the extreme, data logging and other archival applications often require UTC—the international standard for civil time—because it is persistent, existing even *beyond* the lifetime of a particular sensor network.
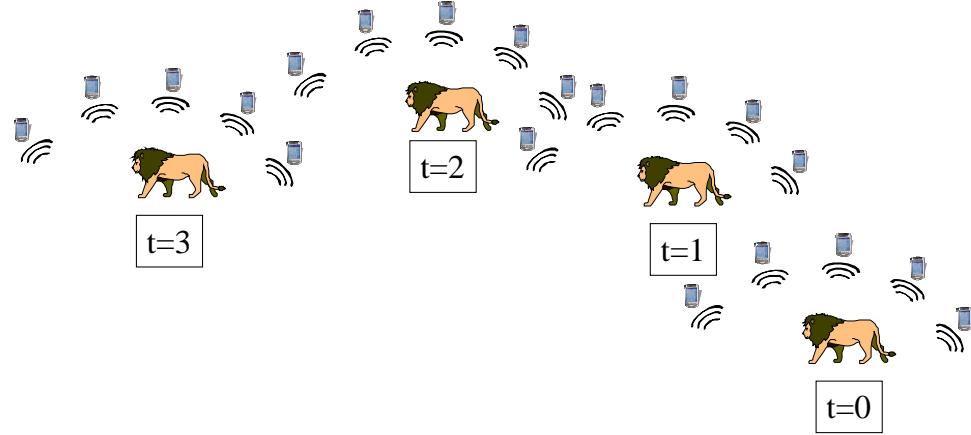
Figure 4.2: A beam-forming array extended to track targets as they move beyond the detection range of a single sensor. Each subset of the array localizes the lion at a single instant by measuring the time of flight of sound: synchronization must have small phase error (microseconds), but only needs a short lifetime (seconds) and small scope (the subset). In contrast, to track the lion's movement through the environment, synchronization must have a longer lifetime (minutes or hours) and a larger scope (network-wide), but the phase error requirements are significantly relaxed (seconds).

4.4   Scope

The *scope* is the size of the region in which a timescale is defined. In some cases, the scope may be purely geographic (e.g., distance in meters). In other contexts, it is more useful to think about the *logical* distance, such as the number of hops through a network. Naturally, these two forms are correlated in a sensor network, scaled by its radios' nominal range.

Our example in Figure 4.2 demonstrates the need for varying scopes, and indeed suggests that scope, lifetime, and phase error are all strongly correlated with each other. A single localization event only requires small-scope synchronization, equal to the acoustic detection range of the sensors. Sensors that can not hear a particular footstep can not participate in the source localization, and thus also do not need to be synchronized. In contrast, to track the long-term behavior of

the lion requires a much larger scope, e.g., equal to the area that the lion covers during its journey through the environment.

Some protocols only synchronize nodes within a single broadcast domain—for example, CesiumSpray, and the synchronization that is part of the 802.11 standard (Section 2.1). But, in general, most network time protocols can synchronize nodes over a large area. As the scope increases, there is usually a cost in cumulative phase error. GPS is a counterexample to this rule—it provides synchronization that is Earth-wide in scope, with the same error bound regardless of location.

The typical correlation of larger scope with larger error dovetails nicely with a natural tendency in sensor networks that comes from their connection to the physical world: tasks over longer timescales and distances tend to tolerate greater absolute error.

4.5   Availability

We use *availability* to mean the completeness of coverage within a method's scope. Some methods may synchronize every node within a region, while others may only synchronize a subset of nodes.

Incomplete availability can arise if a scheme requires special resources that are not uniformly available, such as special hardware, access to infrastructure, or large energy reserves. GPS, for example, is Earth-wide in scope, but its availability is limited to nodes that have GPS receivers. GPS is also not available indoors, under dense foliage, underwater, or during Mars exploration. In contrast, a purely network-based scheme is available to every node that has a radio.

4.6 Internal vs. External

In many distributed systems, network time synchronization simply means, "get the correct time from the network." Of course, this definition implies that a notion of *the* correct time exists. Typically, the time is UTC—the international standard for civil time. However, in sensor networks, UTC not always needed. Indeed, in some cases, the definition of an SI second itself may not be relevant. In other words, there are situations where distributed nodes need to be *synchronized*, but not necessarily running at a particular frequency known *a priori*. For example:

- *Total ordering of events*—some applications need to record a sequence of events in their temporal *order*, but do not need the events' absolute time. In this case, the frequency of clocks is unimportant, as long as their phase offset is the same. Lamport's work (Section 2.7) takes this idea to its extreme by doing away with physical clocks entirely, in favor of virtual clocks that only "tick" when an event occurs.

- *Coordinated actuation*—some applications need to plan for collaborative action. In many cases, the exact time at which the plan is executed is far less important than ensuring that all nodes act simultaneously. Examples include TDMA frame scheduling (Section 3.4) and distributed robotics (Section 3.3).

This class of applications is sometimes said to require *internal* or *relative* synchronization: the network must be internally consistent, but its relationship to outside time standards is not needed or known.

A second class of applications require *external* synchronization—that is, each node is synchronized to an outside timescale such as UTC. For example, this

need can arise in interactions with people, who for the most part wear wrist-watches that are set to UTC (or, an integral number of hours offset from it). To understand a request such as "Show me all targets detected between 2PM and 8PM," the network and the requestor must share a common time frame. External synchronization is also important in applications such as data archival, because the synchronization lifetime must exist beyond the lifetime of the network collecting the data.

Finally, there is a third class of applications: a hybrid of internal and external synchronization. In some cases, nodes need a *frequency* standard, but do not need a standard *time*. That is, some applications must know the *length* of an SI second (as we described in Section 2.5), but do not care *which* second is currently elapsing according to UTC. Applications that measure phenomena in the physical world often fall into this category. For example, acoustic ranging and beamforming applications assume that they know the speed of sound: about 345 meters/second[2]. To convert propagation delay to distance, the delay must be measured using the same definition of a "second" as that which is implicitly part of the speed-of-sound constant. Similarly, motion-tracking applications need the correct definition of a second to report a target's speed in units (meters/second) that are meaningful.

4.7   Energy Budget

We saw in Chapter 1 that the need for energy efficiency pervades virtually all aspects of sensor networks' design. The exact energy constraints are difficult to quantify because there is a wide range of hardware found across the spectrum of

---

[2]The speed of sound in air varies with temperature; it is actually $331.4 + 0.6T_c$ m/s.

network implementations, and often within a single network. Some nodes have large batteries and run all the time; others are so constrained that they only wake up occasionally, take a single sensor reading, and transmit it before immediately returning to sleep.

Synchronization methods are similarly diverse: some require only a single packet; others are very chatty, sending and receiving energy-consuming messages frequently. Some methods require special hardware: GPS receivers, oven-controlled oscillators, and rubidium frequency standards all require extra energy. Even moderate use of the CPU for time synchronization can dramatically change the energy profile of a node if it would otherwise be in an energy-conserving sleep mode.

We do not consider number of bytes sent or channel utilization as a separate metric. Communication is a major consumer of energy, so energy budget is a good proxy metric for channel utilization. In addition, an energy metric can capture aspects of a scheme such as CPU use or the need for special hardware.

4.8   Convergence Time

Some time synchronization methods can reach an answer almost immediately— for example, a (relatively imprecise) phase error estimate can be made after a single packet exchange. Other methods take significantly longer—for example, GPS receivers typically need between 5 and 10 minutes to acquire from "cold start," during which time they download satellite ephemeris data that are gradually broadcast over the GPS signal.

The importance of convergence time as a metric is directly tied to the need for energy efficiency. If energy were not a concern, our chosen form of synchronization

could simply be left running continuously. Though convergence time might be important due to its effect on network startup, it would make no difference in the steady-state. However, in this energy-constrained world, systems are often forced to power down as many services as possible for as long as possible. This makes convergence time important to consider.

In addition to energy use, convergence time is often tied intimately to two other metrics: phase and frequency error. Most algorithms gradually refine the quality of their synchronization over time. As more information is accumulated, outlier rejection becomes easier. Measurement of precise frequency differences is also easier over long timescales.

For example, Figure 4.3 shows the phase error between two nodes being synchronized with NTP. (Our testbed and data collection methodology are described in Section 6.4.) After 9 minutes of data collection, NTP was confident enough in its enough information to make its first frequency adjustment. However, the best steady-state phase accuracy (about 50μsec) was not achieved until the software had run for about an hour.

4.9   Cost and Form Factor

Finally, for synchronization methods that use special hardware, it is important to consider the components' monetary cost and physical size. These factors can fundamentally change the economics of a sensor network, making a scheme a non-starter. For example, it makes little sense to put a $50 GPS receiver or a $500 Rubidium oscillator on a sensor node that would otherwise cost $5. It may be difficult to integrate special hardware of any kind with dust-mote sized nodes [KKP99].
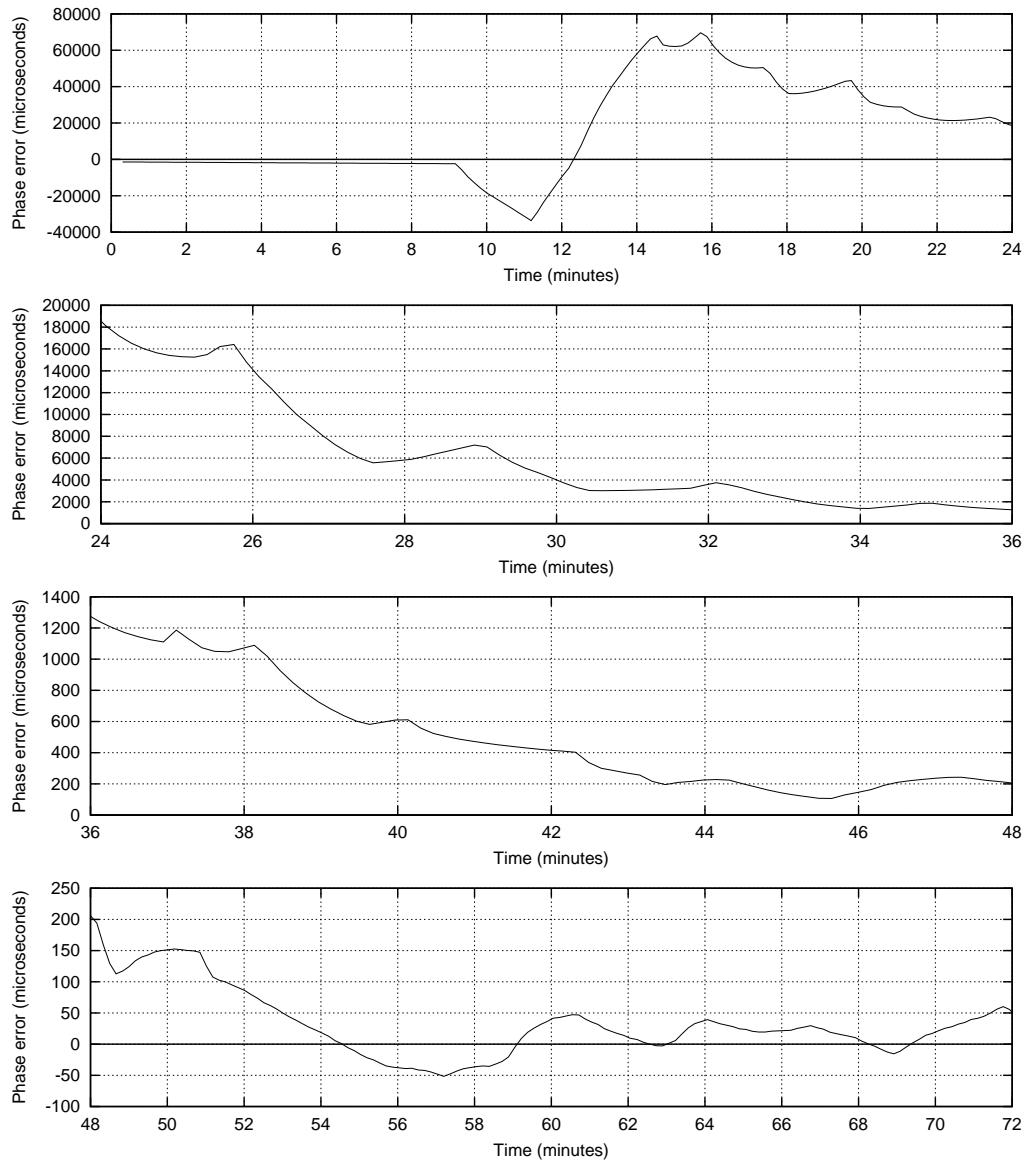
Figure 4.3: The gradual improvement in phase error between two nodes running NTP shows the tradeoff between error and convergence time. These data were collected using two Compaq iPAQ PDAs running Familiar Linux [FAM] connected by Orinoco 802.11 wireless Ethernet cards. Synchronization was provided by NTP version 4.1.74 set to poll every 16 seconds, the maximum rate allowed. Phase error was tested every 10 seconds. NTP was started at $t = 0$, after which it collected data for 9 minutes before acting. During this time, the two nodes drifted apart by about $2\mu\text{sec/sec}$. Peak error was about $70{,}000\mu\text{sec}$ after 16 minutes. Error was reduced to $7{,}000\mu\text{sec}$ after 29 minutes, $700\mu\text{sec}$ after 39 minutes, and $70\mu\text{sec}$ after 53 minutes.

4.10 Summary

In this chapter, we explored a range of metrics that are relevant to time synchronization in sensor networks. We have also seen that these dimensions of time are forever intertwined; it is impossible to optimize one without affecting others. For example:

- It is impossible to correct phase error without introducing temporary frequency error.

- In multi-hop networks, larger-scope synchronization inevitably leads to poorer phase error.

- In single-event forms of synchronization (e.g., a single packet to serve as a reference beacon), phase error grows with lifetime at a rate dictated by frequency error.

- The best phase and frequency error bounds are often achieved when algorithms are given a longer time to converge, but longer convergence time implies more energy use.

- Synchronization methods that offer the best combination of error, scope, lifetime, and availability sometimes require extra hardware that is too large or costly to be viable for a sensor network node.

- . . .

These metrics gives us an important basis for evaluating existing schemes, and designing new ones. In addition, we are now in a position to answer the question posed in earlier chapters: are existing schemes sufficient in the new sensor network regime? Do we really need something new?

CHAPTER 5

A New Regime for Time Synchronization

Time is what we want most, but... what we use worst.

—William Penn

For three decades, Mills' Network Time Protocol [Mil94a] has been the gold standard for network time synchronization. It is the oldest continuously operating protocol in the Internet, and currently serves over 100,000 clients; virtually every server in the world uses it to keep tightly synchronized to UTC [Mil03]. The NTP protocol and its implementation have been battle-tested: they are robust, scalable, and provide excellent service. Why do sensor networks need something new?

The essence of the answer is that sensor networks are, in some ways, a radical departure from traditional networking; they break many of the assumptions on which prior work is based. While existing solutions are critical building blocks, we argue that new solutions are needed. This chapter explores the issue in more detail by revisiting some of the related work we described in Chapter 2. We will compare the service that existing methods such as NTP can provide against the range of requirements seen in Chapter 3, using the metrics we described in Chapter 4.

In this chapter, we also propose a number of general guidelines for the design of new time synchronization systems for sensor networks. As we will see in

later chapters, these design themes are common to many of the synchronization methods we have developed or encountered during our exploration of this problem space.

5.1   The Need for Something New

Computer clock synchronization has seen dramatic improvements in recent times. The current state of the art provides error bounds that are orders of magnitude better than just ten or twenty years prior. It is now common for an ordinary desktop machine to be continuously synchronized to UTC within perhaps 100–200μsec. Such accuracy has been made possible through the confluence of two main technologies: the Global Positioning System (GPS) and the Network Time Protocol (NTP).

As we described in Section 2.6, GPS has emerged as the pre-eminent method for distributing time and frequency world-wide. Today (2003), without Selective Availability, a relatively inexpensive ($500) GPS clock can achieve $<$ 20nsec phase error to UTC [CNS]. It is interesting to note that the level of service provided by modern GPS receivers has, in some sense, turned time synchronization on its head. Years ago, methods for synchronizing to an external clock such as UTC was were much lower precision than methods for keeping a local cluster kept internally consistent. Today, the opposite is true: GPS receivers both provide an external, persistent, global clock *and* can produce higher precision agreement among nodes than most methods of network time synchronization.

There are practical limitations to using GPS, of course: special receiver hardware is required and an antenna must be mounted on a roof or some other location that has a view of the sky. While this is done for certain applications that re-

quire high precision timing, the cost and effort of using GPS is not justified for most desktop users. However, now that the Internet provides ubiquitous access to network time servers, most computers no longer require direct access to GPS.

The second major technological piece, therefore, has been the development of Mills' Network Time Protocol. NTP effectively delivers a *proxy* UTC signal via the Internet to those machines (the majority) that do not have a direct view of GPS time.[1] It has been adopted by the vast majority of users who need synchronization but do not require the sub-microsecond precision that GPS provides. The exact precision achieved by NTP over an IP network depends on the network topology, but in the best cases it can achieve between 50 and 100 μsec, as we will see in Section 6.4.

The combination of GPS and NTP has proved very successful. Both the NTP protocol and its implementation have been battle-tested: they are robust, scalable, and provide high accuracy. Many in the sensor-network research community have asked: "Why not use NTP here, too?" As we saw in Section 2.2, a number of groups have moved in this direction, implementing NTP-like hierarchical time distribution for sensor networks [GKA02, SA02]. But is this truly the best choice? Many of the assumptions that NTP makes, while true in the Internet, are not true in sensor networks. It is important to understand those assumptions and how they are broken.

5.1.1 Energy Aware

As we saw in Chapter 1, finite energy is a crucial constraint in sensor networks. The need for energy efficiency violates a number of assumptions routinely made

---

[1]NTP servers can use sources of UTC other than GPS: the WWVB radio station and GOES satellites, for example. However, GPS is the most common.

by classical synchronization algorithms:

- Using the CPU in moderation is free.

- Listening to the network is free.

- Occasional transmissions have a negligible impact.

These assumptions are true in traditional networks and consequently have become fundamental to many synchronization schemes. For example, NTP assumes that the CPU is always on. As we saw in Figure 4.1, it performs frequency discipline of the oscillator by adding small but continuous offsets to the system clock. In addition, NTP does not attempt to predict the time at which packets will arrive; it simply listens to the network all the time. Finally, while NTP is conservative in its use of bandwidth, it assumes a continuous ability to transmit packets during normal operation. While it does deal with a loss of network access by continuing to discipline the clock using the last known frequency, connectivity loss is assumed to be a rare event.

Pottie and Kaiser articulated why these assumptions do not hold in sensor networks [PK00]. In a low-power radio, receiving a packet requires nearly as much energy as sending one; the RF energy radiated is small compared to the cost of the electronics behind it. Listening to an idle channel while waiting for a packet to arrive requires nearly as much energy as reception of a packet.

Because of the cost of listening, the radio can (and often must) remain powered off until an event of interest occurs. Intermittent connectivity may indeed be the norm, and not a special case. This is a problem for NTP; it can take a long time after network access has been restored before it achieves its optimal accuracy, as we saw in Figure 4.3.

CPU cycles can also be a constrained resource: the limited energy mandates the use of slow processors which spend most of their time powered down. For example, Pottie and Kaiser's original WINS design included a low-power pre-processor that awakens the main processor after an event of interest. A scheme that assumes continuous access to the CPU, even for short intervals, will not work properly if the CPU is completely off most of the time.

5.1.2 Infrastructure-Supported vs. Ad Hoc

Two commonly held perceptions are that NTP is used to synchronize "the entire Internet" and that it can synchronize clocks to a precision of several hundred microseconds. While these statements are individually accurate, the resulting assumption is not: that NTP alone can produce clock agreement that is world-wide in scope *and* has sub-millisecond phase error. When we examine just how Internet-scale time synchronization is achieved, we see that this is not quite true. NTP gets help from something that, often times, is not available in a sensor network: *infrastructure.*

NTP allows construction of time synchronization hierarchies, each rooted at one of many canonical sources of external time in the Internet. The canonical sources ("Stratum 1" servers, in NTP terminology) are synchronized with each other via a variety of "out of band" mechanisms—for example, radio receivers for time signals from the Global Positioning System or the WWVB radio broadcast we described in Sections 2.6 and 5.1. This infrastructure provides a common view of a global timescale (UTC) to the Stratum 1 servers throughout the Internet. Consequently, nodes throughout the Internet enjoy being synchronized to a single, *global* timescale while rarely finding themselves more than a few hops away from a *local* source of this canonical time.

Sensor networks, on the other hand, may often consist of large-diameter networks *without* an external infrastructure. Often it is not an option to equip sensor nodes with receivers for out of band time references—GPS requires line of sight to the satellites, which is not available inside of buildings, beneath dense foliage, underwater, when jammed by an enemy, or during Mars exploration. In addition, in many contexts, GPS receivers are too expensive in terms of their energy consumption, size, or cost. GPS may not be practical, for example, in a network made up entirely of nodes on the scale of a dust-sized mote, or even the Berkeley COTS mote.

In this scenario, NTP-style algorithms must create a hierarchy rooted at a single node that is designated as the system's master clock. For this to work, we must first develop an algorithm that automatically maintains such a hierarchy in the face of node dynamics and partitions. (Partitioning has problems of its own, as we will see in Sections 5.1.3 and 5.1.4.) However, even assuming this problem has been solved, there is still a fundamental stumbling block: *if there a single source of canonical time, most nodes will be far away from it.* Error accumulates at each hop, so nodes that are far away from the master clock will be poorly synchronized to the global timescale.

This is a particularly unfortunate situation in a sensor network, where nodes often need the best synchronization to nearby neighbors—e.g., for distributed acoustic beamforming as we saw in Section 3.6. Consider the scenario shown in Figure 5.1. Nodes A, B, and C are close to one another, but far away from the master clock. Because A and C have different paths to the master, their errors will be uncorrelated. In an NTP-like scheme, B must choose either A or C as its synchronization source. Either choice will lead to poor synchronization when sharing data with the opposite neighbor. For example, if B synchronizes to C,
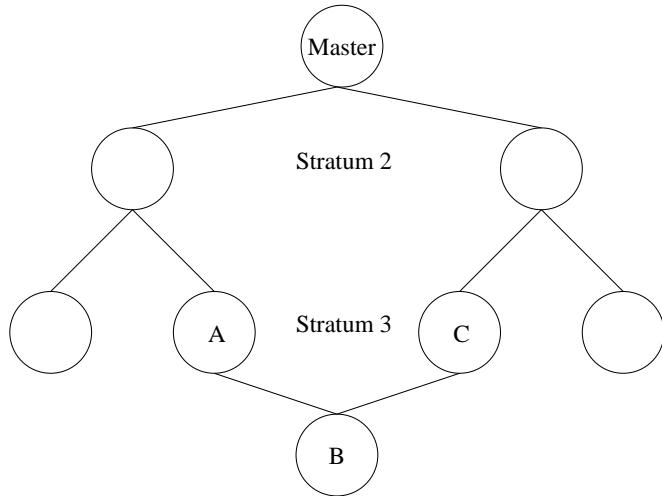
Figure 5.1: A global timescale can lead to poorly synchronized neighbors, if the neighbors are far from the master clock and have uncorrelated loss due to divergent synchronization paths.

its synchronization error to A will be quite large; the synchronization path leads all the way to the master and back. (Alternatively, B can take the average of A and C, in which case it is not particularly well synchronized to either one of them.)

The point of this example is to demonstrate that the quality of the synchronization among the group is proportional to the distance of the group from an arbitrarily selected master node. That node may be far away. Because of the inherently localized computation in sensor networks, we would prefer that a nodes' synchronization be commensurate with their distance from *each other*, not their distance from an unrelated location. As we will discuss in Section 5.2.5, this constraint suggests that sensor networks should have *no global timescale*.

5.1.3 Static Topology vs. Dynamics

Most synchronization algorithms assume that the network topology is relatively stable. Of course, the Internet suffers from transient link failures, but it works most of the time. This leads to an insidious problem: because network outages are considered rare, most schemes do not provide a way to correct the timestamps that were acquired during an outage.

Consider a network that is partitioned into two disconnected regions. Assume a synchronization algorithm such as NTP is running, extended to dynamically elect a master clock (e.g., as described in [GKA02] or [SA02]). Nodes within each region will be *internally* synchronized. However, lacking some form of infrastructure, the two time domains will have no known relationship to one another.

Imagine now that during the network partition, events of interest occur in each region, and are timestamped. Some time later, a change in the topology allows communication between the two regions. This may happen for any number of reasons: perhaps a source of RF interference disappeared, a node moved, or a sleeping node was activated. Even though the entire network can now communicate, it is impossible to compare events that were witnessed in different regions: their timestamps have no relationship to one another.

This problem arises because existing time synchronization schemes generally do not provide a way to correct old timestamps based on new information. It is one symptom of a deep-seated assumption implicitly made by virtually all schemes: that a node's clock is kept synchronized *all the time.* Applications are not concerned with the details of synchronization—how, when, or with which nodes it is performed; they simply expect a "correct" clock to be available at any instant that it is needed. We believe that in sensor networks, this service model is fundamentally and fatally flawed; in Section 5.2.4, we propose an alternative.
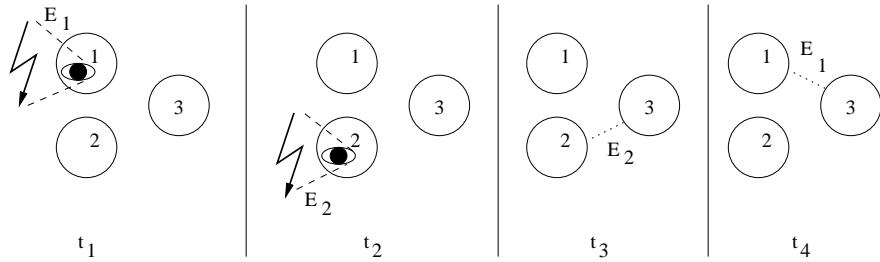
Figure 5.2: A disconnected network that requires time synchronization. Some form of time synchronization is needed to compare the event detections by nodes 1 and 2. However, these devices are never in direct communication, and the indirect communication they share (via node 3) only comes after the events in question. *Figure courtesy of Kay Römer, ETH Zurich. Used with permission.*

5.1.4 Connected vs. Disconnected

In the previous example of a partitioned network, synchronization was needed at a time that a link was not available. An even more extreme condition might sometimes exist—synchronization may be needed between nodes that *never* communicate.

Consider the scenario depicted in Figure 5.2, in which three nodes (1, 2, and 3) are collaboratively monitoring pollution in a river. At time t_1 (*left*), Node 1 detects a oil stain. Node 2 detects the same phenomenon at t_2. At t_3, Node 2 is close enough to Node 3 that it can inform Node 3 of the detection event (E_2). Finally, at t_4, Node 1 informs Node 3 of E_1.

The times of E_1 and E_2 must be compared according to a common timebase; however, Node 1 and Node 2 have never exchanged information that could have been used to synchronize their clocks. As in the example from Section 5.1.3, this situation will confound traditional synchronization algorithms that assume nodes must be synchronized *a priori*, before they detect events of interest.

5.1.5 Single-Hop vs. Multi-Hop

Although NTP synchronizes over multi-hop networks, much of the other, non-Internet ("LAN") work in distributed clock agreement assumes that all nodes in the system can directly exchange messages. In other words, they assume a single latency and jitter bound is common to all messages in the system. Some methods that exploit the broadcast property of the physical media [MFN00, VRC97] do not speak to the problem of federating the clocks of multiple (overlapping) broadcast domains.

Sensor networks span many hops; the end-to-end latency is much larger than a single hop. The required synchronization scope is often significantly larger than a broadcast domain. This makes it difficult to apply methods that assume a fully connected or common-latency topology.

5.2 Design Principles

Having described the shortcomings of traditional time synchronization schemes in the previous section, we can begin to formulate new directions for time synchronization in sensor networks. In this section, we present general design themes that have been culled from both our own work (Chapters 6–9) and the experiences of others in the area (e.g., Römer [R01, ER02], Su [SA02] and Ganeriwal [GKA02]).

Of course, our design guidelines have not yet withstood the test of time in the way that NTP has for the Internet. However, they are a reasonable starting point for thinking about how time synchronization might meet all the special requirements in this domain: energy efficiency, scalability, robustness, ad-hoc deployment, and so on.

5.2.1 Multi-Modal Synchronization

As we saw in Chapter 2, there are many different ways to achieve time synchronization. Even within the new sensor network domain, many methods have been proposed. These schemes all fall into many disparate points in the parameter space we described in Section 4, including energy, phase and frequency error, scope, availability, lifetime, cost, and convergence time. Every scheme makes trade-offs—no single method is optimal along all axes. For example:

- Typical GPS receivers access a timescale that is persistent and Earth-wide in scope to a precision of 20ns or better [CNS]. However, receivers require several minutes of settling time, and may be too large, costly, or high-power to justify on a small sensor node. In addition, the GPS infrastructure is not always available (§5.1.2).

- Römer's scheme described in [R01] achieves 1ms precision, creates an instantaneous timescale with little overhead, and works on unidirectional links. However, the synchronization is localized and rather short-lived.

- Our RBS scheme (Chapter 6) can achieve several μsec of phase error, and sufficient frequency estimates to extend the timescale for several minutes. RBS synchronizes all nodes within a broadcast domain. However, it requires a bidirectional broadcast medium and several packet exchanges.

- The multihop extension to RBS described in Chapter 7 allows the timescale to be extended across multiple broadcast domains, but at the cost of (somewhat) degraded accuracy.

None of these methods can be considered the *best*; each has advantages and disadvantages. The details of a particular application and hardware will dictate
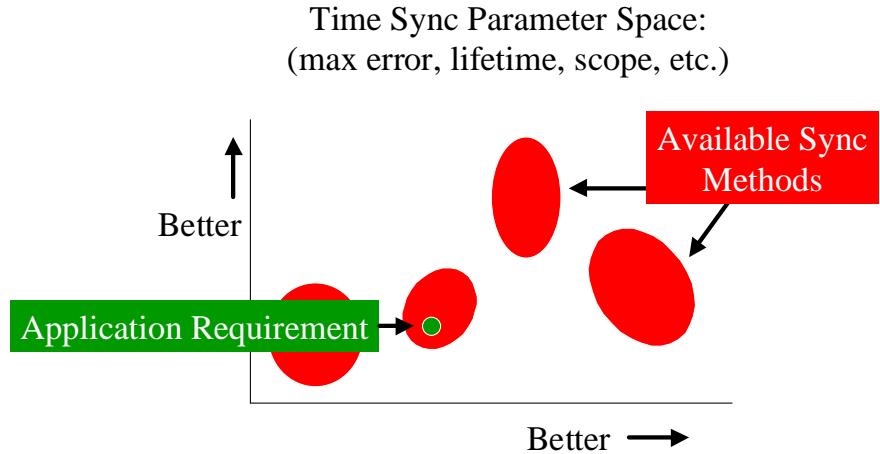
Figure 5.3: Sensor network designers need a palette of synchronization methods that fall into a diverse set of points in the parameter space. This makes it more likely that a method will be necessary and sufficient for an application. The distance between the requirements and the method represents waste—resources such as energy that are being expended to provide service that the application doesn't need.

the method that should be used in each situation.

Ideally, we would like to have a large enough palette of methods available so that we can choose an overall scheme that is both *necessary and sufficient* for the application on all axes, as seen in Figure 5.3. Unnecessary synchronization wastes resources; insufficient synchronization leads to poor application performance.

5.2.2 Tuning Parameters

Most synchronization methods have flexibility in the way they operate—they can make trade-offs that improve one metric at the expense of another. For example, a scheme might send a larger number of packets, trading energy use for lower phase and frequency error bounds. Setting radios to their maximum transmit power can trade energy for scope.

These kinds of parameters should be exposed and easily changeable by the ap-

plication designer user so that the service provided matches the requirements as closely as possible. Without tunability, the synchronization methods depicted in Figure 5.3 would be points. The more tuning parameters, the larger the region—making it more likely that at least one region will precisely overlap the application requirement. Distance from the requirement to the synchronization method, if any, represents waste—resources such as energy that are being expended to provide service that the application doesn't need.

5.2.3 Tiered Architectures

Although Moore's law predicts that hardware for sensor networks will inexorably become smaller, cheaper, and more powerful, technological advances will never prevent the need to make trade-offs. Even as our notions of metrics such as "fast" and "small" evolve, there will always be compromises: nodes will need to be faster *or* more energy-efficient, smaller *or* more capable, cheaper *or* more durable.

Instead of choosing a single hardware platform that makes a particular set of compromises, we believe an effective design is one that uses a tiered platform consisting of a heterogeneous collection of hardware. Larger, faster, and more expensive hardware (beacons, aggregation points) can be used more effectively by also using smaller, cheaper, and more limited nodes (sensors, tags). An analogy can be made to the memory hierarchy commonly found in desktop computer systems. CPUs typically have extremely expensive, fast on-chip cache, backed by slower but larger L2 cache, main memory, and ultimately on-disk swap space. This organization, combined with a tendency in computation for locality of reference, results in a memory system that appears to be as large and as cheap (per-byte) as the swap space, but as fast as the on-chip cache memory.

In sensor networks, where localized algorithms are a primary design goal, similar benefits can be realized by creating the network from a spectrum of hardware ranging from small, cheap, and numerous, to large, expensive, and powerful. For example, if at least *one* of the nodes in a sensor network can be equipped with GPS, this can help enormously in applications that require an external time reference (e.g., for data logging). Of course, nodes many hops away from a single GPS-equipped node will not have optimal precision to the UTC timescale.

5.2.4 Explicit Timestamp Conversion

In Section 5.1.3, we described an important problem that arises when using traditional synchronization methods in sensor networks: event timestamps are useless if an event occurs at a time when the clock is not synchronized. This is certainly a problem if outages are common, but it also points to a more fundamental issue: *the usual synchronization service model implies that the clock must be synchronized all the time.* Applications simply assume a synchronized clock is always available—that they can ask "What time is it?" at any time, and receive a correct answer. (Ignore for a moment the thorny question of what "correct" means—we address this in the next section.)

The traditional service model is easy to understand; asking "What time is it?" matches our experience in the real world. However, it is enormously limiting: it forces the synchronization service to make a set of assumptions about the way in which applications use time that are, at best, compromises. We have found a new interface to be far more more flexible, based around the following two questions:

- What time is it *here*?

- At time t here, what time is it *there*?

In other words, we suggest that every node keep time *according to its own clock*. A node should never set its clock or discipline its frequency, but rather let it run at its natural rate. "What time is it here?" simply returns the value of the local clock. It is always available. Time synchronization schemes—regardless of the underlying method—don't set the clock, but rather build a set of parameters relating the phase and frequency of the local clock to other timescales in the system. The remote timescales may be other nodes' clocks, standard timescales or frequency references such as UTC, average timescales that represent a federation of clocks, or even virtual timescales (alá Lamport) that represent event ordering. The second of our two service-model questions can be answered using these parameters: timestamps from different nodes can be compared by using the conversion parameters to transform timestamps into the same time-base.

This service model disconnects time-stamping from time-synchronization, and has many advantages. For example:

- The clock is freed from the requirement that it must be synchronized continuously. Instead, a node can query the clock for a timestamp when an event occurs, and perform synchronization after the fact. This idea of *post-facto synchronization* is described in more detail in Section 5.2.6, and an implementation is demonstrated in Chapter 8.

- Intermittent and disconnected operation (as described in Sections 5.1.3 and 5.1.4) is handled much more naturally. In the old service model, events must be discarded if they occur at a time when synchronization is not available. Or, even worse, erroneous data may be used if the application is not aware of the current (incorrect) state of the clock. In the new service model, timestamps can be collected at the time of the event, then converted whenever sufficient synchronization data are available. The conversion might be pos-

sible immediately, or might not be possible until later due to a temporary service outage. If conversion can not be performed, it fails explicitly.

- The clock is no longer forced to keep a single definition of the "correct" definition of time. Explicit conversion allows the application to specify *which* timescale is needed, which frees the system from needing a global timescale. We will explore this idea in more detail in Section 5.2.5.

- The local clock runs undisciplined, and thus is monotonic and relatively stable—a critical feature to many signal processing algorithms. While frequency drift will occur due to the oscillator's instability (affected by temperature, shock, and voltage variations), there will be no sudden changes in the frequency or phase due to new information arriving at a disciplining algorithm.

- An undisciplined clock requires no continuous corrections to the clock by the CPU or kernel, as are required by algorithms such as NTP. This is important for energy conservation, as we saw in Section 5.1.1.

5.2.5 No Global Timescale

We have seen that many different timescales exist in a sensor network that might be of interest to a node, depending on its task. For example, in some cases, a node may need access to UTC, but can tolerate high error (e.g., general debug messages). In other cases, UTC is irrelevant, but the node needs very high precision agreement with other nodes in the area (e.g., acoustic ranging). However, as we saw in Section 5.1.2, schemes that keep all nodes synchronized to a single, global timescale can lead to unwanted compromises. Unless multiple canonical sources of that timescale are available throughout the network, most nodes will

be far away from it. Phase error between nearby nodes can be proportional to the nodes' distance from an arbitrary master clock, rather than their distance from each other.

By separating local timestamping from timestamp conversions, as we described in the previous section, we also make it easy to perform *per-neighbor* conversions. That is, there is *no global timescale.* The situation shown in Figure 5.1 is no longer a problem—B can simply use its A conversion parameters when communicating with A, and its C parameters when communicating with C. In fact, time conversion can be built into the packet forwarding mechanism itself. That is, nodes can perform successive time conversions on packets as they are forwarded from node to node—keeping timestamps with respect to the local clock at each hop.

5.2.6 Post-Facto Synchronization

As we discussed in earlier sections, most traditional schemes keep the clock synchronized at all times so that it can be consulted if an event occurs. Among other problems, this is wasteful of energy: if events occur infrequently, nodes waste the resources that are used to maintain synchronization during idle periods.

With the traditional service model, little can be done about this situation, except perhaps to predict when synchronization will be needed in the future. However, using our model of local clocks and explicit conversions, a far superior solution falls out quite naturally: *post-facto synchronization.* That is, a node's local clock runs, unsynchronized, at its natural rate, and is used to timestamp events when they occur. After an event of interest, a synchronization process can be started, allowing timestamps from distributed nodes to be reconciled. Waste is reduced because the system avoids performing synchronization at times when

it's not necessary. This scheme works much better than the predictive version: it is much easier to characterize the past than it is to predict the future.

In Chapter 8, we will discuss post-facto synchronization in more detail and show several experiments with a real implementation.

5.2.7 Exploitation of Domain Knowledge

Much of the design of the Internet—and, in fact, the Internet Protocol (IP) itself—is meant to put a consistent interface on top of a heterogeneous and inconsistent tapestry of underlying transport media and protocols. NTP shares a similar philosophy: it makes a reasonable set of "lowest common denominator" assumptions about the environment in which it expects to operate. In the Internet, this is the right choice: it has allowed NTP to become deployed nearly ubiquitously, despite the wide variety of processors, oscillators, network media, node topologies, and cross-traffic it encounters.

The disadvantage of such a design is that it precludes the system from taking advantage of any special features that might be available. In a sensor network, where every possible resource must often be squeezed from the system, it may not be feasible to give up performance for the sake of generality. It often makes sense for each application to take advantage of whatever special features are available at every layer of the system.

For example, the inherent properties of some communication media can be leveraged for time synchronization. In networks that provide a physical-layer broadcast channel, we have shown Reference-Broadcast Synchronization (RBS) can achieve much better precision than NTP by exploiting the properties inherent to broadcast messages (Chapter 6). In time-division based MAC layers, some form of synchronization already exists between radios, and can often be accessed

by a synchronization process on the CPU [Rub79]. Some radio standards such as Bluetooth [BLU] provide a separate synchronous communication channel with low delay jitter, which can be used for exchanging synchronization pulses.

Time synchronization can also use domain knowledge about the application. For example, Römer's scheme [R01] piggybacks round trip time measurements to ordinary data packets sent by other processes. This achieves time synchronization without imposing any additional load on the network. Similarly, RBS can work by observing extant broadcasts in the system instead of sending its own special packets.

5.3   Summary

Many forms of time synchronization have been developed over the years. But, in some ways, sensor networks are a radical departure from traditional networking; they break many of the assumptions on which prior work is based. While existing solutions are critical building blocks, we have seen a number of important issues that sensor networks bring to bear, requiring many aspects of traditional designs to change. Unlike the Internet, sensor networks must be synchronized with an eye toward energy efficiency; they must often work without infrastructure; timescales of mutually disconnected nodes must be reconciled.

Based on both our own work and the experience of others designing in this problem space, we have proposed a number of guidelines that are useful in formulating new synchronization schemes for sensor networks. Perhaps foremost on this list is that there can never be a *single* scheme: each makes different trade-offs, and none are optimal on all axes. We can gain enormous flexibility in designing a palette of such schemes by changing the service interface of time

synchronization—making timestamping and timestamp synchronization separate and explicit steps. The lack of infrastructure can be embraced by doing away with the idea of a global timescale.

Using these guidelines, we have designed, implemented, and characterized a number of new synchronization schemes that have proved themselves very useful in sensor networks. In the chapters to come, we turn to a examination of those schemes in greater detail.

CHAPTER 6

Reference Broadcast Synchronization

If you liked this broadcast, we hope you'll watch it again tomorrow night, and maybe tell your neighbors about it.

—Dan Rather on the CBS Evening News, November 29, 1999

Many network synchronization algorithms have been proposed over the years, but most share the same basic design: a server periodically sends a message containing its current clock value to a client. In this chapter, we explore a form of time synchronization that differs from the traditional model. The fundamental property of our design is that it *synchronizes a set of receivers with one another*, as opposed to traditional protocols in which senders synchronize with receivers. In our scheme, nodes periodically send a message to their neighbors using the network's physical-layer broadcast. Recipients use the message's arrival time as a point of reference for comparing their clocks. The message contains no explicit timestamp, nor is it important exactly when it is sent. We call this *Reference-Broadcast Synchronization*, or RBS.

In this chapter, we use measurements from two wireless implementations to show that removing the sender's nondeterminism from the critical path in this way results in a dramatic improvement in synchronization over using NTP. Using off-the-shelf 802.11 wireless Ethernet, RBS can produce high-precision clock agreement ($1.85 \pm 1.28\mu$sec), while using minimal energy. We also show that this

is a significant improvement over NTP when tested under similar conditions.

The most significant limitation of RBS is that it requires a network with a physical broadcast channel. It can not be used, for example, in networks that employ point-to-point links. However, it is applicable for a wide range of applications in both wired and wireless networks where a broadcast domain exists and higher-precision or lower-energy synchronization is required than what NTP can typically provide.

The organization of this chapter is as follows. In Section 6.1, we describe the design of traditional time synchronization protocols in more detail, and contrast it to the RBS design philosophy. In Section 6.2, we describe the basic building blocks of RBS, including estimation of phase offset (§6.2.1) and clock skew (§6.2.2). We then present empirical data from two wireless implementations of single-hop RBS—on Berkeley motes (§6.3) and on a commercial-off-the-shelf platform based on an 802.11 wireless Ethernet network (§6.4). Finally, in Section 6.5, we summarize the advantages of RBS.

6.1 Traditional Synchronization Methods

Before discussing RBS in detail, we describe how existing time synchronization protocols typically work, and their usual sources of error.

Many synchronization protocols exist, and most share a basic design: a server periodically sends a message containing its current clock value to a client. A simple one-way message suffices if the typical latency from server to client is small compared to the desired accuracy. A common extension is to use a client request followed by a server's response. By measuring the total round-trip-time of the two packets, the client can estimate the one-way latency. This allows for more

accurate synchronization by accounting for the time that elapses between the server's creation of a timestamp and the client's reception of it. NTP [Mil94a] is a ubiquitously adopted protocol for Internet time synchronization that exemplifies this design.

6.1.1   Sources of Time Synchronization Error

The enemy of precise network time synchronization is *non-determinism*. Latency estimates are confounded by random events that lead to asymmetric round-trip message delivery delays; this contributes directly to synchronization error. To better understand the source of these errors, it is useful to decompose the source of a message's latency. Kopetz and Schwabl characterize it as having four distinct components [KS89]:

- *Send Time*—the time spent at the sender to construct the message. This includes kernel protocol processing and variable delays introduced by the operating system, e.g. context switches and system call overhead incurred by the synchronization application. Send time also accounts for the time required to transfer the message from the host to its network interface.

- *Access Time*—delay incurred waiting for access to the transmit channel. This is specific to the MAC protocol in use. Contention-based MACs (e.g., Ethernet [MB83]) must wait for the channel to be clear before transmitting, and retransmit in the case of a collision. Wireless RTS/CTS schemes such as those in 802.11 networks [ISO99] require an exchange of control packets before data can be transmitted. TDMA channels [Rub79] require the sender to wait for its slot before transmitting.

- *Propagation Time*—the time needed for the message to transit from sender to receivers once it has left the sender. When the sender and receiver share access to the same physical media (e.g., neighbors in an ad-hoc wireless network, or on a LAN), this time is very small as it is simply the physical propagation time of the message through the media. In contrast, Propagation Time dominates the delay in wide-area networks, where it includes the queuing and switching delay at each router as the message transits through the network.

- *Receive Time*—processing required for the receiver's network interface to receive the message from the channel and notify the host of its arrival. This is typically the time required for the network interface to generate a message reception signal. If the arrival time is timestamped at a low enough level in the host's operating system kernel (e.g., inside of the network driver's interrupt handler), the Receive Time does not include the overhead of system calls, context switches, or even the transfer of the message from the network interface to the host.

Existing time synchronization algorithms vary primarily in their methods for estimating and correcting for these sources of error. For example, Cristian observed that performing larger numbers of request/response experiments will make it more likely that at least one trial will not encounter random delays [Cri89]. This trial, if it occurs, is easily identifiable as the shortest round-trip time. NTP filters its data using a variant of this heuristic.

Our scheme takes a different approach to reducing error. Instead of estimating error, we exploit the broadcast channel available in many physical-layer networks to remove as much of it as possible from the critical path. Our contributions in this paper stem from the observation that a message that is broadcast at the
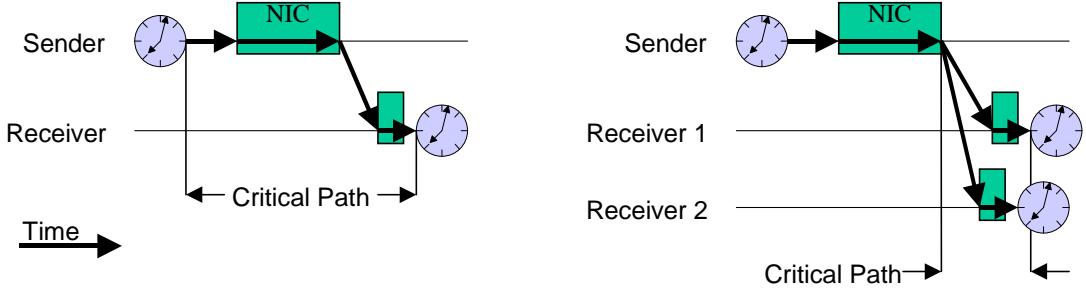
Figure 6.1: A critical path analysis for traditional time synchronization protocols (*left*) and RBS (*right*). For traditional protocols working on a LAN, the largest contributions to nondeterministic latency are the Send Time (from the sender's clock read to delivery of the packet to its NIC, including protocol processing) and Access Time (the delay in the NIC until the channel becomes free). The Receive Time tends to be much smaller than the Send Time because the clock can be read at interrupt time, before protocol processing. In RBS, the critical path length is shortened to include only the time from the injection of the packet into the channel to the last clock read.

physical layer will arrive at a set of receivers with very little variability in its delay. Although the Send Time and Access Time may be unknown, and highly variable from message to message, the nature of a broadcast dictates that for a *particular* message, these quantities are *the same for all receivers*. This observation was also made by Veríssimo and Rodrigues [VR92], and later became central to their CesiumSpray system [VRC97].

The fundamental property of Reference-Broadcast Synchronization is that a broadcast message is only used to *synchronize a set of receivers with one another*, in contrast with traditional protocols that synchronize the sender of a message with its receiver. Doing so removes the Send Time and Access Time from the critical path, as shown in Figure 6.1. This is a significant advantage for synchronization on a LAN, where the Send Time and Access time are typically the biggest contributors to the nondeterminism in the latency. Mills attributes most of the phase error seen when synchronizing an NTP client workstation to a GPS

receiver on the same LAN (500μsec–2000μsec in his 1994 study [Mil94b]) to these factors—Ethernet jitter and collisions.

To counteract these effects, an RBS broadcast is always used as a *relative* time reference, never to communicate an absolute time value. It is exactly this property that eliminates error introduced by the Send Time and Access Time: each receiver is synchronizing to a reference packet which was, by definition, injected into the physical channel at the same instant. The message itself does not contain a timestamp generated by the sender, nor is it important exactly when it is sent. In fact, the broadcast does not even need to be a dedicated timesync packet. Almost any extant broadcast can be used to recover timing information—for example, ARP packets in Ethernet, or the broadcasted control traffic in wireless networks (e.g., RTS/CTS exchanges or route discovery packets).

6.1.2 Characterizing the Receiver Error

Previously, we described the four sources of latency in sending a message. Two of these—the Send Time and Access Time—are dependent on factors such as the instantaneous load on the sender's CPU and the network. This makes them the most nondeterministic and difficult to estimate. As described above, RBS eliminates the effect of these error sources altogether; the two remaining factors are the Propagation Time and Receive Time.

For our purposes, we consider the Propagation Time to be effectively 0. The propagation speed of electromagnetic signals through air is close to c,[1] and through copper wire about $\frac{2}{3}c$. For a LAN or ad-hoc network spanning tens of feet, propagation time is at most tens of nanoseconds, which does not contribute significantly to our μsec-scale error budget. (In sensor networks, the error budget

---

[1]A convenient rule of thumb is $1nsec$/foot.

is often driven by the need to sense phenomena, such as sound, that move much more slowly than the RF-pulse used to synchronize the observers.) In addition, RBS is only sensitive to the *difference* in propagation time between a pair of receivers, as depicted in Figure 6.1.

The length of a bit gives us an idea of the Receive Time. For a receiver to interpret a message at all, it must be synchronized to the incoming message within one bit time. Latency due to processing in the receiver electronics is irrelevant as long as it is *deterministic*, since our scheme is only sensitive to *differences* in the receive time of messages within a set of receivers.[2] In addition, the system clock can easily be read at interrupt time when a packet is received; this removes delays due to receiver protocol processing, context switches, and interface-to-host transfer from the critical path.

To verify our assumptions about expected receiver error, we turned to our laboratory's wireless sensor network testbed [CEE01]. Specifically, we tested the Berkeley Mote, a postage-stamp sized narrowband radio and sensor platform developed by Pister *et al.* at Berkeley [KKP99]. Motes use a minimal event-based operating system developed by Hill *et al.* specifically for that hardware platform called TinyOS [HSW00]. We programmed 5 Motes to raise a GPIO pin high upon each packet arrival, and attached those signal outputs to an external logic analyzer that recorded the time of the packet reception events. An additional Mote emitted 160 broadcast packets over the course of 3 minutes, with random inter-packet delays ranging from 200msec to 2 seconds. For each pulse transmitted, we computed the difference in the pulse's packet reception time for each of the 10 possible pairings of the 5 receivers. Some pulses were lost at some receivers due to the normal vagaries of RF links. A total of 1,478 pairings were computed.

---

[2]It is important to consider effects of any intentional nondeterminism on the part of a receiver, such as aggregation of packets in a queue before raising an interrupt.
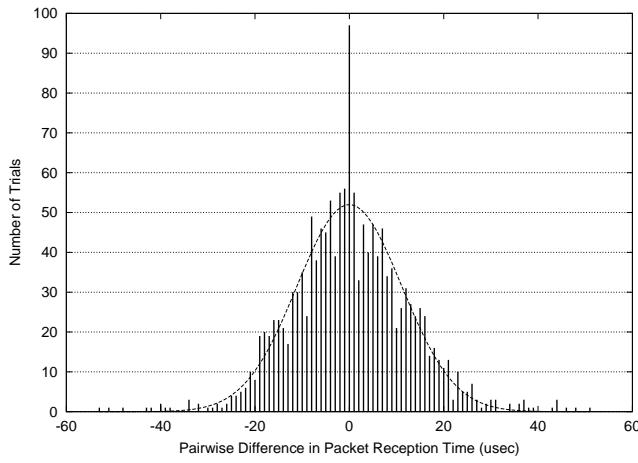
Figure 6.2: A histogram showing the distribution of inter-receiver phase offsets recorded for 1,478 broadcast packets, grouped into 1μsec buckets. The curve is a plot of the best-fit Gaussian parameters. ($\mu = 0, \sigma = 11.1\mu$sec, confidence=99.8%)

A histogram showing the distribution of the receivers' phase offsets is shown in Figure 6.2. The maximum phase error observed in any trial was 53.4μsec. The Mote's radios operate at 19,200 baud and thus have a bit time of 52μsec. This correspondence supports our reasoning that a receiver's jitter is unlikely to be much greater than a single bit time, as long as the receiver electronics have a deterministic delay between packet reception and host notification.

The phase offset distribution appears Gaussian; the chi-squared test indicates 99.8% confidence in the best fit parameters—$\mu = 0, \sigma = 11.1\mu$sec. This is useful for several reasons. As we will see in later sections, a well-behaved distribution enables us to significantly improve on the error bound statistically, by sending multiple broadcast packets. In addition, Gaussian receivers are easy to model realistically in numeric analyses. This facilitates quick experimentation with many different algorithms or scenarios whose systematic exploration is impractical, such as examining the effects of scale.

6.2 Reference-Broadcast Synchronization

So far, we characterized the determinism of our receiver hardware. We now turn to the question of using that receiver to achieve the best possible clock synchronization.

We should note that our research does not encompass the question of how to build a more deterministic receiver. This is best answered by those who design them. Plans for forthcoming revisions of the Mote's TinyOS are reported to include better phase lock between a receiver and incoming message. Instead, we ask: How well can we synchronize clocks with a *given* receiver? How is the precision we can achieve related to that receiver's jitter?

In this section, we build up the basic RBS algorithm for nodes within a single broadcast domain. (Applying RBS in multihop networks will be considered in Section 7.) First, we consider how RBS can be used to estimate the relative phase offsets among a group of receivers. Next, we describe a slightly more complex scheme that also corrects for clock skew. We also describe implementation of RBS on both Berkeley Motes and commodity hardware, and quantify the performance of RBS vs. NTP.

6.2.1 Estimation of Phase Offset

The simplest form of RBS is the broadcast of a single pulse to two receivers, allowing them to estimate their relative phase offsets. That is:

1. A transmitter broadcasts a reference packet to two receivers (i and j).

2. Each receiver records the time that the reference was received, according to its local clock.

3. The receivers exchange their observations.

[For simplicity, we will not delve into the details of the protocol here. Instead, we will simply assume that all receivers can exchange observations with each other, so as to focus on the essence of the algorithm. However, it is important to note that this assumption does bring with it an unwanted topology constraint: receivers must be able to hear reference broadcasts *and* be within transmission range of each other. In our real RBS implementations, this constraint does not exist—receivers report observations of a broadcast back to the node that sent it. This scheme reduces the complexity of information exchange, and only requires that links be bi-directional.]

Based on a single broadcast, the receivers have sufficient information to form a local (or *relative*) timescale. Global (or *absolute*) timescales such as UTC are important, and we will see in Section 7.4 how RBS can be extended to provide them. However, in this section we will first consider construction of local timescales— which are often sufficient for sensor network applications. For example, if node i at position $(0, 0)$ detects a target at time $t = 4$, and j at position $(0, 10)$ detects it at $t = 5$, we can conclude that the target is moving north at 10 units per second. This comparison does not require absolute time synchronization; the reference broadcast makes it possible to compare the time of an event observed by i with one observed by j.[3]

The performance of this scheme is closely related to three factors:

1. The number of receivers that we wish to synchronize (n). We assumed above that $n = 2$; however, collaborative applications often require syn-

---

[3]Our prototype implementation never sets the local clock based on reference broadcasts; instead, it provides a library function for timestamp conversion. That is, it can convert a time value that was generated by the local clock to the value that would have been generated on another node's clock at the same time, and vice-versa.

chronization of many nodes simultaneously. As the set grows, the likelihood increases that at least one will be poorly synchronized. We define the *group dispersion* as the maximum phase error between any of the $\binom{n}{2}$ pairs of receivers in a group.

2. The nondeterminism of the underlying receiver, as we discussed in Section 6.1.2.

3. Clock skew in the receivers, as their oscillators all tick at different rates. A single reference will provide only *instantaneous* synchronization. Immediately afterward, the synchronization will degrade as the clocks drift apart. Precision will therefore decay as more time elapses between the synchronization pulse and the event of interest. Typical crystal oscillators are accurate on the order of one part in 10^4 to 10^6 [Vig92]—that is, two nodes' clocks will drift 1–100μsec per second.

We will see in the next section how clock skew can be estimated. However, assume for the moment that we already know the clock skew and have corrected for it. Let us instead consider what is required to correct for the nondeterminism in the receiver. Recall from Figure 6.2 that we empirically observed the Mote's receiver error to be Gaussian. We can therefore increase the precision of synchronization statistically, by sending more than one reference:

1. A transmitter broadcasts m reference packets.

2. Each of the n receivers records the time that the reference was observed, according to its local clock.

3. The receivers exchange their observations.

4. Each receiver i can compute its phase offset to any other receiver j as the average of the phase offsets implied by each pulse received by both nodes i and j. That is, given

$$n: \text{the number of receivers}$$

$$m: \text{the number of reference broadcasts, and}$$

$$T_{r,b}: r\text{'s clock when it received broadcast } b,$$

$$\forall i \in n, j \in n: \text{Offset}[i,j] = \frac{1}{m} \sum_{k=1}^{m} (T_{j,k} - T_{i,k}). \qquad (6.1)$$

Because this basic scheme does not yet account for clock skew (a feature added in §6.2.2), it is not yet implementable on real hardware. We therefore turned to a numeric simulation based on the receiver model that we derived empirically in Section 6.1.2. In each trial, n nodes were given random clock offsets. m pulse transmission times were selected at random. Each pulse was "delivered" to every receiver by timestamping it using the receiver's clock, including a random error matching the Gaussian distribution parameters shown in Figure 6.2 ($\sigma = 11.1$). An offset matrix was computed as given in Equation 6.1. Each of these $O(n^2)$ computed offsets was then compared with the "real" offsets; the maximum difference was considered to be the *group dispersion*. We performed this analysis for values of $m = 1 \ldots 50$ and $n = 2 \ldots 20$. At each value of m and n, we performed 1,000 trials and calculated the mean group dispersion, and standard deviation from the mean. The results are shown in Figure 6.3.

This numeric simulation suggests that in the simplest case of 2 receivers (the lower curve of Figure 6.3a), 30 reference broadcasts can improve the precision
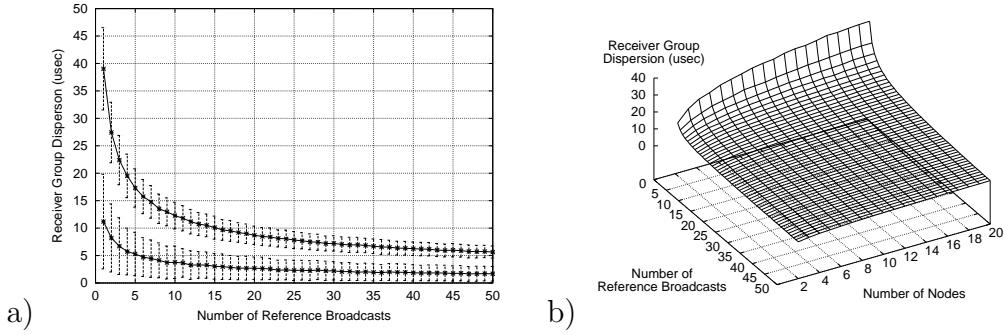
Figure 6.3: Analysis of expected group dispersion (i.e., maximum pairwise error) after Reference-Broadcast Synchronization. Each point represents the average of 1,000 simulated trials. The underlying receiver is assumed to report packet arrivals with error conforming to the distribution in Figure 6.2. *a)* Mean group dispersion, and standard deviation from the mean, for a 2-receiver (*bottom*) and 20-receiver (*top*) group. *b)* A 3D view of the same dataset, from 2 to 20 receivers (inclusive).

from 11μsec to 1.6μsec, after which we reach a point of diminishing return.[4] In a group of 20 receivers, the dispersion can be reduced to 5.6μsec. Figure 6.3b shows a 3D view of the same dataset for all receiver group sizes from 2 to 20. This view also shows that larger numbers of broadcasts significantly mitigate the effect of larger group size on precision decay.

6.2.2 Estimation of Clock Skew

So far, we have described a method for estimating phase offset assuming that there was no clock skew—that is, that all the nodes' clocks are running at the same rate. Of course, this is not a realistic assumption. A complete description of crystal oscillator behavior is beyond the scope of this paper;[5] however, to a

---

[4]11μsec is the expected average result for a single pulse delivered to two receivers; this is the standard deviation of the inter-receiver phase error, found in Section 6.1.2, upon which the analysis was based.

[5][Vig92] has a good introduction to the topic and pointers to a more comprehensive bibliography.

first approximation, the important characteristics of an oscillator are:

- Accuracy—the agreement between an oscillator's expected and actual frequencies. The difference is the *frequency error*; its maximum is usually specified by the manufacturer. The crystal oscillators found in most consumer electronics are accurate to one part in 10^4 to 10^6.

- Stability—An oscillator's tendency to stay at the same frequency over time. Frequency stability can be further subdivided into *short-term* and *long-term* frequency stability. Short-term instability is primarily due to environmental factors, such as variations in temperature, supply voltage, and shock. Long-term instability results from more subtle effects, such as oscillator aging.

The phase difference between two nodes' clocks will change over time due to frequency differences in the oscillators. The basic algorithm that we described earlier is not physically realizable without an extension that takes this into account—in the time needed to observe 30 pulses, the phase error between the clocks will have changed.

Complex disciplines exist that can lock an oscillator's phase and frequency to an external standard [Mil98]. However, we selected a very simple yet effective algorithm to correct skew. Instead of *averaging* the phase offsets from multiple observations, as shown in Equation 6.1, we perform a *least-squares linear regression.* This offers a fast, closed-form method for finding the best fit line through the phase error observations over time. The frequency and phase of the local node's clock with respect to the remote node can be recovered from the slope and intercept of the line.

Of course, fitting a *line* to these data implicitly assumes that the frequency is stable, i.e., that the phase error is changing at a constant rate. As we mentioned

earlier, the frequency of real oscillators will change over time due, for example, to environmental effects. In general, network time synchronization algorithms (e.g., NTP) correct a clock's phase and its oscillator's frequency error, but do not try to model its frequency instability. That is, frequency adjustments are made continuously based on a recent window of observations relating the local oscillator to a reference. Our prototype RBS system is similar; it models oscillators as having high short-term frequency stability by ignoring data that is more than a few minutes old.

6.3 Implementation on Berkeley Motes

To test these algorithms, we first built a prototype RBS system based around the Berkeley Motes we described in Section 6.1.2. In the first configuration we tested, 5 Motes periodically broadcasted a reference pulse with a sequence number. Each of them used a 2μsec resolution clock to timestamp the reception times of incoming broadcasts. We then performed an off-line analysis of the data. Figure 6.4a shows the results of a typical experiment after automatic rejection of outliers (described below). Each point is the difference between the times that the two nodes reported receiving a reference broadcast, plotted on a timescale defined by one node's clock. That is, given

$$T_{r,b}: r\text{'s clock when it received broadcast } b,$$

for each pulse k that was received by receivers r_1 and r_2, we plot

$$x = T_{r_1,k}$$
$$y = T_{r_2,k} - T_{r_1,k}$$
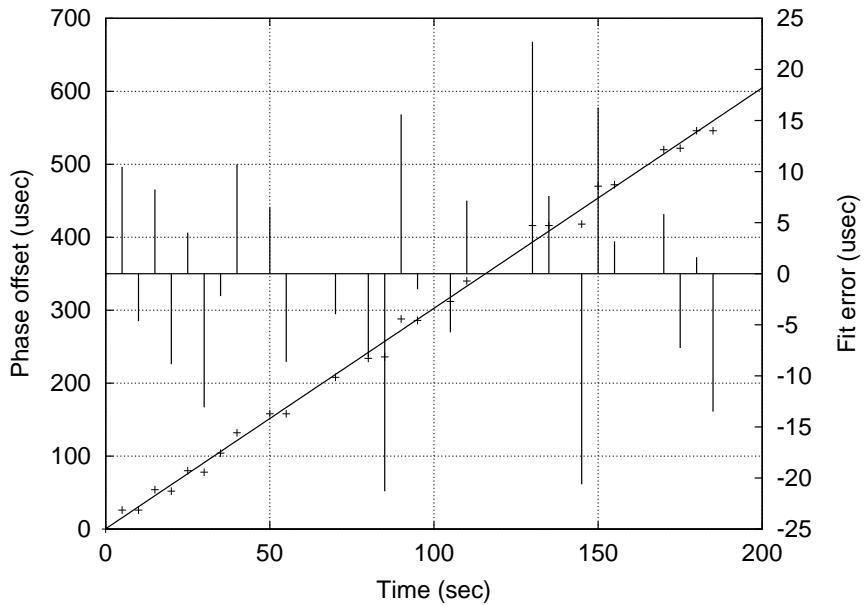
Figure 6.4: An analysis of clock skew's effect on RBS. Each point represents the phase offset between two nodes as implied by the value of their clocks after receiving a reference broadcast. A node can compute a least-squared-error fit to these observations (*diagonal lines*), and thus convert time values between its own clock and that of its peer. *a*) Synchronization of the Mote's internal clock. The vertical impulses (read with respect to the *y2* axis) show the distance of each point from the best-fit line.

For visualization purposes, the data are normalized so that the first pulse occurs at $(0,0)$. The diagonal line drawn through the points represents the best linear fit to the data—i.e., the line that minimizes the RMS error. The vertical impulses, read with respect to the right-hand y axis, show the distance from each point to the best-fit line.

The residual error—that is, the RMS distance of each point to the fit line— gives us a good idea of the quality of the fit. We can reject outliers by discarding the point that contributes most to the total error, if its error is greater than some threshold (e.g. 3σ). In this experiment, the RMS error was 11.2μsec. (A more systematic study of RBS synchronization error is described in later sections.)

The slope of the best-fit line defines the clock skew, when measured by receiver r_1's clock. We can see, for example, that these two oscillators drifted about 300μsec after 100 "r_1 seconds." The line's intercept defines the initial phase offset. When combined, we have sufficient information to convert any time value generated by r_1's clock to the value that would have been generated by r_2's clock at the same time. We will see in Section 7.4 that "r_2" may also be an external standard such as UTC. However, even the internally consistent mapping is useful in a sensor network, as we saw in Section 6.2.1.

6.4   Commodity Hardware Implementation

The performance of RBS on Berkeley Motes was very encouraging. However, it is reasonable to wonder if its success was due to the algorithm itself or simply the fact that it was implemented on a tightly integrated radio and processor platform. In addition, we were curious about the relative performance of RBS and NTP. To answer these questions—and provide RBS on a platform more gener-

ally available to users—we implemented RBS as a UNIX daemon, using UDP datagrams instead of Motes. We used a commodity hardware platform that is part of our testbed: StrongARM-based Compaq IPAQs with Lucent Technologies 11Mbit 802.11 wireless Ethernet adapters. Our IPAQs run the "Familiar" Linux distribution [FAM] with Linux kernel v2.4. In this test, all the wireless Ethernet adapters are connected to a wireless 802.11 base station.

To make the comparison as fair as possible, we first implemented RBS under the same constraints as NTP: a pure userspace application with no special kernel or hardware support, or special knowledge of the MAC layer (other than that it supports broadcasts). Like NTP, our Linux RBS daemon uses UDP datagrams sent and received via the Berkeley socket interface. Packet reception times are recorded in userspace by using the standard gettimeofday() library function (and underlying system call). The daemon records the time after learning that a new packet has arrived via an unblocked select() call. The IPAQ's clock resolution is 1μsec.

Our RBS daemon simultaneously acts in both "sender" and "receiver" roles. Every 10 seconds (slightly randomized to avoid unintended synchronization), each daemon emits a pulse packet with a sequence number and sender ID. The daemon also watches for such packets to arrive; it timestamps them and periodically sends a report of these timestamps back to the pulse sender along with its receiver ID. The pulse sender collects all of the pulse reception reports and computes clock conversion parameters between each pair of nodes that heard its broadcasts. These parameters are then broadcast back to local neighbors. The RBS daemons that receive these parameters make them available to users. RBS never sets the nodes' clocks, but rather provides a user library that converts UNIX timevals from one node ID to another.

The clock conversion parameters are the slope and intercept of the least-square linear regression, similar to the examples shown in Figure 6.4. Each regression is based on a window of the 30 most recent pulse reception reports. In practice, a small number of pulses are outliers, not within the Gaussian mode shown in Figure 6.2, due to random events such as concurrent serial and Ethernet interrupts. These outliers are automatically rejected based on an adaptive threshold equal to 3 times the median fit error of the set of points not yet rejected. (Early versions used a more traditional "3σ" approach, but the standard deviation was found to be too sensitive to gross outliers.) The remaining points are iteratively re-fit, and the outlier threshold recomputed, until no further points are rejected. If more than half the points are rejected as outliers, the fit fails.

To test the synchronization accuracy, we connected a GPIO output from each of two IPAQs to an external logic analyzer. The analyzer was programmed to report the time difference between two pulses seen on each of its input channels. We wrote a Linux kernel module that accepts a target pulse-time from a userspace application, then emits a GPIO pulse when the local clock indicates that the target time has arrived. By implementing the pulsing service inside the kernel, with interrupts disabled shortly before the target time arrives, there is virtually no phase error between the GPIO pulse and the node's clock. In other words, the kernel module ensures that error between the pulses as seen by the logic analyzer is entirely due to the nodes' clock synchronization error, not an artifact of the test. (This kernel module purely facilitates testing; it was not used to help the synchronization of either NTP or RBS.)

Using this experimental framework, we tested three different synchronization schemes:

- RBS—Our reference broadcast synchronization. A third IPAQ was used as

a pulse sender to facilitate synchronization between the two test systems. NTP was off during this test.

- NTP—The standard NTP package, version 4.1.1a, ported to our IPAQs. The ntpdate program was first used to achieve gross synchronization. Then, the ntpd daemon was run, configured to send synchronization packets every 16 seconds (the maximum frequency it allows). The clients were allowed to synchronize to our lab's stratum 1 GPS-steered NTP server for several hours at this high sampling frequency before the test began. The NTP server was on the wired side of our network (i.e., accessed via the wireless base station).

- NTP-Offset—In our test, RBS has a potentially unfair advantage over NTP. Although NTP maintains a continuous estimate of the local clock's phase error with respect to its reference clock, it also limits the rate at which it is willing to correct this error. This is meant to prevent user applications from seeing large frequency excursions or discontinuities in the timescale. Our initial RBS implementation has no such restriction. Since our test only evaluates phase error, and does not give any weight to frequency stability, it might favor RBS. To eliminate this possible advantage, we created *NTP-Offset* synchronization. This was similar to the NTP method; however, during each trial, the test application queried the NTP daemon (using the ntpq program) for its current estimate of the local clock's phase error. This value was subtracted from the test pulse time.

For each of these synchronization schemes, we tested two different traffic scenarios:

- Light network load—The 802.11 network had very little background traffic

other than the minimal load generated by the synchronization scheme itself.

- Heavy network load—Two additional IPAQs were configured as traffic generators. Each IPAQ sent a series of randomly-sized UDP datagrams, each picked uniformly between 500 and 15,000 bytes (IP fragmentation being required for larger packets). The inter-packet delay was 10msec. The cross-traffic was entirely among third parties—that is, the source and destination of this traffic were neither the synchronization servers nor the systems under test. The average aggregate offered load of this cross-traffic was approximately 6.5Mbit/sec.

Each of the six test scenarios consisted of 300 trials, with an 8 second delay between each trial, for a total test time of 40 minutes per scenario. The results are shown in Figures 6.5 and 6.6.

In the light traffic scenario, RBS performed more than 8 times better than NTP—an average of 6.29 ± 6.45 μsec error, compared to $51.18 \pm 53.30\mu$sec for NTP. 95% of RBS trials had an error of 20.53μsec or better, compared to a 131.20μsec bound for 95% of NTP trials.

Much of the 6.29μsec error seen with our userspace RBS implementation is due to the jitter in software—the code path through the network stack, the time required to schedule the daemon, and the system calls from userspace to determine the current time. We will see in the next section that significantly better precision can be achieved using packet timestamps acquired inside the kernel's network interrupt handler.

In our heavy traffic scenario, the difference was even more dramatic. As we discussed in Section 6.1, NTP is most adversely affected by path asymmetries that confound its latency estimate. When the network is under heavy load, it

Synchronization Error with Light Network Load

Method	Mean Error	Std Dev	50%	95%	99%
RBS	6.29	6.45	4.22	20.53	29.61
NTP	51.18	53.30	42.52	131.20	313.64
NTP-Offset	204.17	599.44	48.15	827.42	4334.50
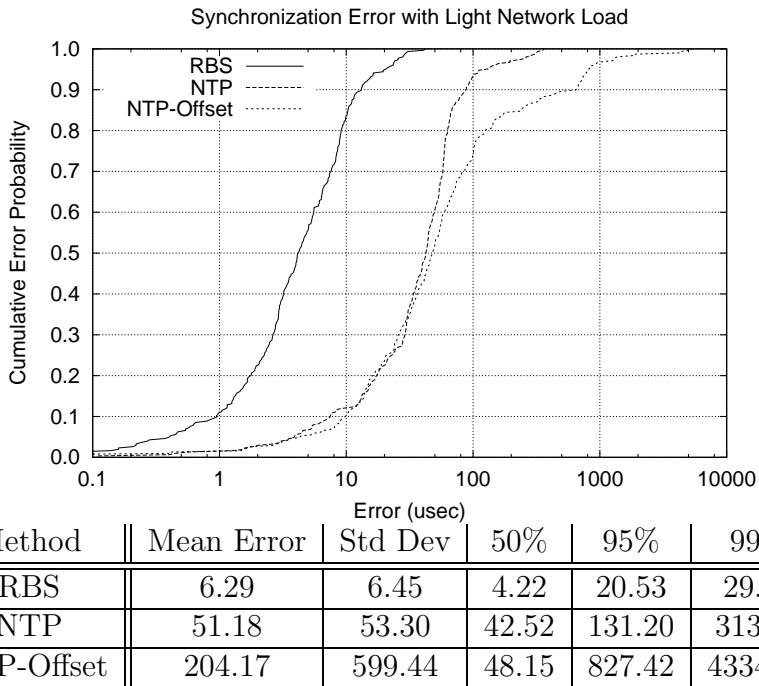
Figure 6.5: Synchronization error for RBS and NTP between two Compaq IPAQs using an 11Mbit 802.11 network. Clock resolution was 1μsec. All units are μsec. "NTP-Offset" uses an NTP-disciplined clock with a correction based on NTP's instantaneous estimate of its phase error; unexpectedly, this correction led to poorer synchronization. RBS performed more than 8 times better than NTP on a lightly loaded network.

Synchronization Error with Heavy Network Load
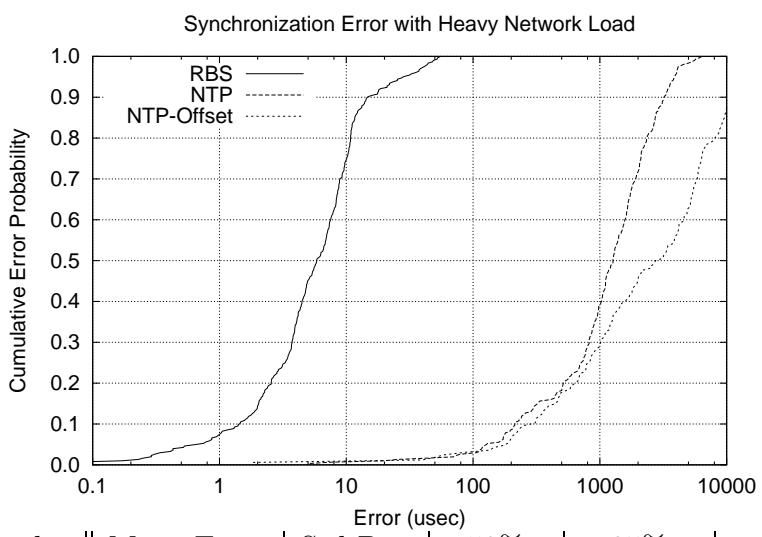
Method	Mean Error	Std Dev	50%	95%	99%
RBS	8.44	9.37	5.86	28.53	48.61
NTP	1542.27	1192.53	1271.38	3888.79	5577.82
NTP-Offset	5139.08	6994.58	3163.11	15156.44	38897.36

Figure 6.6: A test similar to that in 6.5, but in the presence of cross-traffic with an average aggregate offered load of approximately 6.5Mbit/sec. On this heavily loaded network, NTP further degraded by more than a factor of 30, while RBS was virtually unaffected.

103

becomes far more likely that the medium access delay will be different during the trip to and from the NTP server. RBS, in contrast, has no dependence on the exact time packets are sent; variable MAC delays have no effect. While NTP suffered more than a 30-fold degradation in precision (average 1,542μsec, 95% within 3,889μsec), RBS was almost completely unaffected (average 8.44μsec, 95% within 28.53μsec). The slight degradation in RBS performance was due to packet loss, which reduced the number of broadcast pulses available for comparison with peers. Several instances of RBS phase excursion were also observed when a packet containing a clock parameter update was lost, forcing clients to continue using aging data.

Surprisingly, the NTP-Offset method almost uniformly performed more poorly than plain NTP. The cause was not clear, but we suspect this was due to the inter-packet jitter in the 802.11 MAC. The low-pass filter built into NTP's clock discipline algorithm was better suited to model the high-jitter network than the more responsive online phase estimator.

6.4.1 RBS using 802.11 and Kernel Timestamps

In the previous section, we described the performance of RBS when implemented as a userspace application. This provided a fair comparison with NTP. However, for practical use in our testbed, we give RBS the additional advantage of packet timestamps acquired in the network interface's interrupt handler. Timestamping at interrupt-time, before the packet is even transferred from the NIC, significantly reduces jitter and is a standard feature of the Linux kernel. The metadata is accessible by reading packets using the LBNL packet capture library, libpcap [MJ93], instead of the usual socket interface.

Using kernel timestamps, the performance of RBS improved considerably, to

1.85 ± 1.28μsec (see Figure 7.3), from 6.29 ± 6.45μsec in the user-space implementation.

We believe this result was primarily limited by the 1μsec clock resolution of the IPAQ, and that finer precision can be achieved with a finer-resolution clock. In our future work, we hope to try RBS on a such a platform: for example, using the Pentium's TSC, a counter which runs at the frequency of the processor core (e.g., 1GHz).

6.5   Summary

In this chapter, we have built up the basic RBS algorithm, showing how reference broadcasts can be used to relate a set of receivers' clocks to one another. Over time, we can estimate both relative phase and skew. On Berkeley Motes, RBS can synchronize clocks within 11μsec, with our relatively slow 19,200 baud radios. On commodity hardware—Compaq IPAQs using an 11 Mbit/sec 802.11 network— we achieved synchronization of 6.29 ± 6.45μsec, 8 times better than NTP under the same conditions. Using kernel timestamps, precision nearly reached the clock resolution—1.85 ± 1.28μsec.

The advantages of RBS include:

- The largest sources of nondeterministic latency can be removed from the critical path by using the broadcast channel to synchronize receivers with one another. This results in significantly better precision synchronization than algorithms that measure round-trip delay.

- Multiple broadcasts allow tighter synchronization because residual errors tend to follow well-behaved distributions. In addition, multiple broadcasts allow estimation of clock skew, and thus extrapolation of past phase offsets.

This enables *post-facto synchronization*, saving energy in applications that need synchronized time infrequently and unpredictably, as we will see in Section 8.2.

- Outliers and lost packets are handled gracefully; the best fit line in Figure 6.4 can be drawn even if some points are missing.

- RBS allows nodes to construct *local* timescales. This is useful for sensor networks or other applications that need synchronized time but may not have an absolute time reference available. Absolute time synchronization can also be achieved, as we will describe in Section 7.4.

The primary shortcoming of RBS as we have described it thus far is that it can only be used to synchronize a set of nodes that lie within a single physical-layer broadcast domain. We address this limitation in the next chapter.

CHAPTER 7

Multi-Hop RBS

If your time to you // Is worth savin' // Then you better start swim-min' // Or you'll sink like a stone // For the times they are a-changin'.

—Bob Dylan

RBS, as we have described it until now, has used broadcasts to coordinate a set of receivers that lie within a single broadcast domain. A natural question that arises is how this technique might be extended so as to coordinate receivers whose span is larger than a single physical-layer broadcast. For example, the extent of wireless sensor networks is usually much larger than the broadcast range of any single node. In traditional LANs, the broadcast domain of an Ethernet is limited; LANs are interconnected with routers, bridges, or other gateways that do not propagate broadcasts at the physical layer.

In these scenarios, an extension to basic RBS can be used to synchronize a group of nodes that lie beyond the range of a single broadcast. Consider the example topology shown in Figure 7.1. The lettered nodes, A and B, both send a sync pulse. A and B can not hear each other, but each of them are heard by 4 receivers. Receivers that are in the same neighborhood (i.e., have heard the same sync pulse) can relate their clocks to each other, as described in the previous chapter. However, notice that receiver 4 is in a unique position: it can hear the
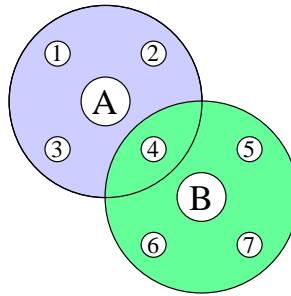
Figure 7.1: A simple topology where multi-hop time synchronization is required. Nodes A and B send sync pulses, establishing two locally synchronized neighborhoods. Receiver 4 hears both A and B, and can relate nodes in either neighborhood to each other.

sync pulses from both A *and* B. This allows receiver 4 to relate the clocks in one neighborhood to clocks in the other.

7.1   Multihop Clock Conversion

To illustrate how the conversion works, imagine that we wish to compare the times of two events that occurred at opposite ends of the network—near receivers 1 and 7. Nodes A and B both send synchronization pulses, at times P_A and P_B. Say that receiver 1 observes event E_1 2 seconds after hearing A's pulse (e.g., $E_1 = P_A + 2$). Meanwhile, receiver 7 observes event E_7 at time $P_B - 4$. These nodes could consult with receiver 4 to learn that A's pulse occurred 10 seconds after B's pulse: $P_A = P_B + 10$. Given this set of constraints:

$$E_1 = P_A + 2$$

$$E_7 = P_B - 4$$

$$P_A = P_B + 10$$

$$\implies E_1 = E_7 + 16$$
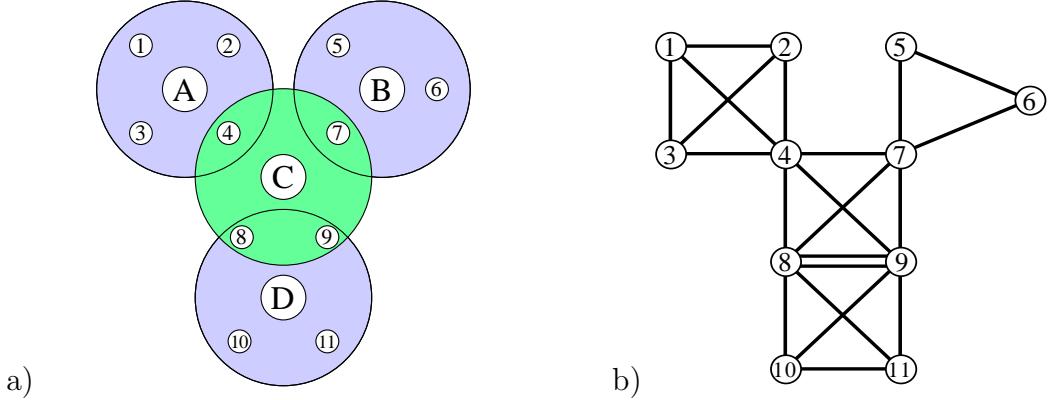
Figure 7.2: A more complex (3-hop) multihop network topology. *a)* The physical topology. Beacons (lettered nodes), whose transmit radii are indicated by the larger shaded areas, broadcast references to their neighbors (numbered nodes). *b)* The logical topology. Edges are drawn between nodes that have received a common broadcast, and therefore have enough information to relate their clocks to each other. Receivers 8 and 9 share two logical links because they have two receptions in common (from C and D).

In this example, it was possible to establish a global timescale for both events because there was an intersection between A's and B's receivers.

This technique for propagating timing information has many of the same benefits as in the single-hop case. A traditional solution to the multi-hop problem might be for nodes to re-broadcast a timing message that they have previously received. In contrast, our technique *never depends on the temporal relationship between a sender and a receiver*. As in the single-hop case, removing all the nondeterministic senders from the critical path reduces the error accumulated at each hop, as we will show in Section 7.3.

For simplicity, the example above spoke of sending "a pulse." Naturally, the phase and skew relationship between receivers in a neighborhood can be determined more precisely by using a larger number of pulses, as we described in Section 6.2. To take advantage of this information, our RBS prototype does not literally compare pulse reception times, as suggested by the constraints in

the equations above. Instead, it performs a series of timebase conversions using the best-fit lines that relate logically-adjacent receivers to each other. Adopting the notation $E_i(R_j)$ to mean the time of event i according to receiver j's clock, we can state the multihop algorithm more precisely:

1. Receivers R_1 and R_7 observe events at times $E_1(R_1)$ and $E_7(R_7)$, respectively.

2. R_4 uses A's reference broadcasts to establish the best-fit line (as in Figure 6.4a) needed for converting clock values from R_1 to R_4. This line is used to convert $E_1(R_1)$ to $E_1(R_4)$.

3. R_4 similarly uses B's broadcasts to convert $E_1(R_4)$ to $E_1(R_7)$.

4. The time elapsed between the events is computed as $E_1(R_7) - E_7(R_7)$.

This algorithm is conceptually the same as the simpler version. However, each timebase conversion implicitly includes a correction for skew in all three nodes.

7.2   Time Routing in Multihop Networks

In Figure 7.1, there was a single "gateway" node that we designated for converting timestamps from one neighborhood's timebase to another. However, in practice, no such designation is necessary, or even desirable. It is straightforward to compute a "time route"—that is, dynamically determine a series of nodes through which time can be converted to reach a desired final timebase.

Consider the network topology in Figure 7.2a. As in Figure 7.1, the lettered nodes represent sync pulse senders; numbered nodes are receivers. (This distinction between senders and receiver roles is purely illustrative; in reality, a node can

play both roles.) Each pulse sender creates a neighborhood of nodes that have known phase offsets by virtue of having received pulses from a common source. These relationships are visualized in Figure 7.2b, in which nodes with known clock relationships are joined by graph edges. A path through this graph represents a series of timebase conversions. For example, we can compare the time of $E_1(R_1)$ with $E_{10}(R_{10})$ by transforming $E_1(R_1) \rightarrow E_1(R_4) \rightarrow E_1(R_8) \rightarrow E_1(R_{10})$.

It is possible to find a series of conversions automatically, simply by performing a shortest-path search in a graph such as in Figure 7.2b. In addition, the weights of the graph edges can be used to represent the quality of the conversion—for example, the residual error (RMS) of the linear fit to the broadcast observations. A minimum-error conversion between any two nodes can be found using a weighted-shortest-path search such as Dijkstra's or Bellman-Ford.

Of course, such shortest-path algorithms do not scale to large networks due to their dependence on global information. Although there is precedent for such systems (e.g., the Internet's early link-state routing algorithms), a more localized approach is needed if the method is to scale. To this end, there is an interesting alternative: time conversion can be built into the extant packet forwarding mechanism. That is, nodes can perform successive time conversions on packets as they are forwarded from node to node—keeping the time with respect to the local clock at each hop. This technique, also suggested by Röemer [R01], is attractive because it requires only local computation and communication. Instead of flooding clock conversion parameters globally, they can be distributed using a tightly scoped broadcast. In addition, the quality of time synchronization across each link can be incorporated into the link metric used by the routing algorithm, ensuring that routing is not completely orthogonal to the quality of the time conversions available.

7.3 Measured Performance of Multihop RBS

In a multihop topology where a series of timestamp conversions are required, as described in the previous section, each conversion step can introduce synchronization error. We now consider the cumulative effects of such errors as the path length grows.

We saw in Section 6.1.2 that the synchronization error introduced by a reference broadcast is Gaussian. In addition, each of the per-hop contributions to the error are *independent* because the phase and skew estimates at each hop are based on a different set of broadcasts. Therefore, we expect that total path error—a sum of independent Gaussian variables—will also follow a Gaussian distribution. If the average per-hop error along the path is σ, then we expect[1] the average path error for an n-hop path to be $\sigma\sqrt{n}$. This bound is pleasing in that it grows slowly, and implies that we can synchronize nodes across many hops without significant decay in precision.

We tested the precision of multihop RBS using the IPAQ and 802.11-based testbed we described in Section 6.4, including the in-kernel packet timestamping discussed in Section 6.4.1. We configured 9 IPAQs in a linear topology, alternating between 5 (numbered) receivers and 4 (lettered) beacon emitters, as shown in Figure 7.3. Each receiver could hear only the nearest sender on its left and right. The test procedure was the same as we described in Section 6.4—in 300 trials, two IPAQs were commanded to raise a GPIO pin high at the same time. A logic analyzer reported the actual time differences. We tested a 1-hop, 2-hop, 3-hop and 4-hop conversion, by testing the rise-time difference between nodes 1–2, 1–3, 1–4, and 1–5, respectively.

---

[1]From the variance's identity that $var(x + y) = var(x) + var(y)$; variance=$\sigma^2$.

Hops	Mean Error	Std Dev	50%	95%	99%
1	1.85	1.28	1.60	4.51	5.85
2	2.73	1.91	2.41	6.49	7.74
3	2.73	2.42	2.22	6.93	9.92
4	3.68	2.57	3.18	8.26	10.70
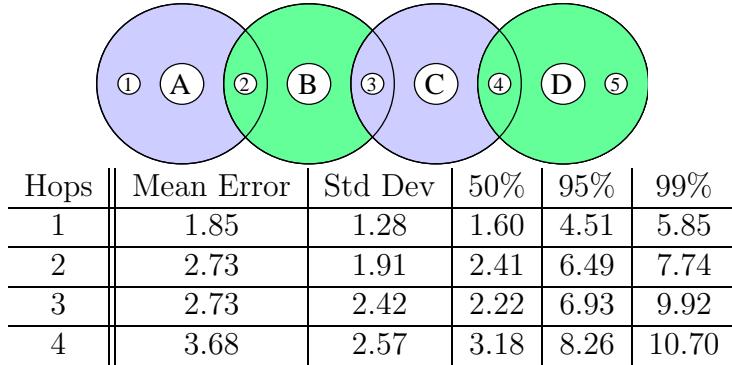
Figure 7.3: *top)* The topology used to test cumulative error when RBS is used in a multihop network. *bottom)* Measured performance of multihop RBS, using kernel timestamping, on IPAQs using 11 MBit/sec 802.11. All units are μsec.

The results, shown in Figure 7.3, generally seem to follow our $\sigma\sqrt{n}$ estimate. The average 4-hop error was $3.68 \pm 2.57\mu$sec, which is nearly $\sigma\sqrt{4}$, normalizing σ to be the average 1-hop error.

7.4 Synchronization with External Timescales

So far, we have spoken entirely of creating local, internally consistent timescales. Although relative synchronization is sufficient for many applications, others require absolute time as measured by an external reference such as UTC. Reference timescales are typically distributed via governmentally sponsored radio systems such as the short-wave WWVB station [Bee81] or the satellite-based Global Positioning System [Kap96]. Commercially available receivers for these systems can synchronize computers by providing them with a "PPS" (pulse-per-second) signal at the beginning of each second. The seconds are numbered out of band, such as through a serial port.

RBS provides a natural and precise way to synchronize a network with such an external timescale. For example, consider a GPS radio receiver connected to

one of the nodes in a multihop network such as the one in Figure 7.2. GPS time simply becomes another node in that network, attached via one edge to its host node. The PPS output of the receiver can be treated exactly as normal reference broadcasts are. That is, the host node timestamps each PPS pulse and compares its own timestamp with the pulse's true GPS time. The phase and skew of the host node's clock relative to GPS time can then be recovered using the algorithms described in Section 6.2. Other nodes can use GPS time by finding a multihop conversion route to it through the host node, using the algorithm we described in Section 7.2.

7.5   RBS in Multihop Wired Networks

Our work is focused on using RBS in wireless sensor networks. However, its use is not limited to that context—it can work on any network which supports a physical-layer broadcast, such as our 802.11 implementation described in Section 6.4. Can the RBS multihop algorithm be used in wired networks?

Consider the simple example topology in Figure 7.4, which depicts two LANs connected by a router. RBS can be used to synchronize all of the end-stations in this network if both LANs are capable of a physical layer broadcast and the router participates in the protocol. The situation is equivalent to the wireless multihop scenario depicted in Figure 7.1, with the router playing the part of Receiver 4. Specifically, the router can record the reception times of broadcast packets it receives from end-stations on both interfaces according to its local clock, and periodically publish its observations back to stations on both sides. (These publications may, themselves, be broadcast.) Queueing delay on the router would not affect RBS because packets only queue while waiting for *outgoing* interfaces; the critical path in RBS ends when the router records the reception time of a
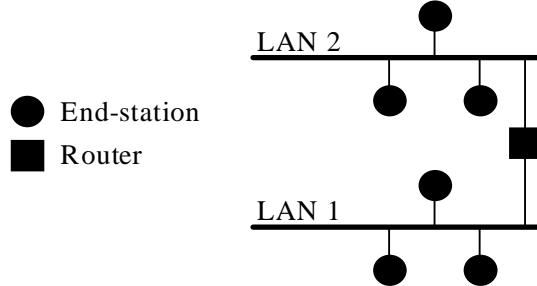
Figure 7.4: Two LANs connected by a router. RBS can be used to synchronize all the end-stations in this network if each LAN supports broadcast and the router participates in the protocol. The router must observe the reception times of broadcasts on both interfaces, and publish that information to the end-stations on both sides.

broadcast on an *incoming* interface.

RBS is not appropriate in the Internet, which consists of point-to-point links. (The required router modifications also make it a non-starter.) However, our future work will include an implementation of the external timescale reference described in the previous section. We believe that it is possible to use RBS to keep workstations within a laboratory—or even a campus—synchronized to UTC within $1\mu s$.

7.6   Summary

In this chapter, we have presented an extension to the basic RBS scheme that allows locally coordinated timescales to be federated into a global timescale, across broadcast domains. Our scheme preserves the fundamental property of RBS: that synchronization is only based on the relationship of one receiver to another. As with single-hop RBS, the multi-hop scheme avoids any dependance on the exact time at which a message is sent. As a result, precision decays slowly—the average error for an n-hop network is proportional to $\sqrt{n}$. In our test of kernel-assisted

RBS across a 4-hop topology, average synchronization error was $3.68 \pm 2.57 \mu\text{sec}$. RBS-federated timescales may also be referenced to an external standard such as UTC if at least one node has access to a source of reference time.

CHAPTER 8

Post-Facto Synchronization

The past cannot be changed. The future is yet in your power.

—Hugh White

To save energy in a sensor network, it is a desirable to keep nodes in a low-power state, if not turned off completely, for as long as possible. Sensor network hardware is often designed with this goal in mind; processors have various sleep modes or are capable of powering down high-energy peripherals when not in use.

This type of design is exemplified by the WINS platform [ADL98], which has a low-power "pre-processor" that is capable of rudimentary signal processing. Normally, the entire node is powered down except for the pre-processor. When the pre-processor detects a potentially interesting signal, it powers on the general purpose processor for further analysis. The CPU, in turn, can power on the node's radio if it determines that an event has occurred that needs to be reported.

Such designs allow the components that consume the most energy to be powered for the least time, but also pose significant problems if we wish to keep synchronized time. Traditional methods try to keep the clock disciplined at all times so that an accurate timestamp is always available. What happens if the radio—our external source of time and frequency standards—is continuously off for hours at a time? Or, in the case of a platform like WINS, what if the general-purpose processor that knows how to discipline the clock is also off?

117

Traditional synchronization schemes make the assumption that the clock must be synchronized *before* the event occurs, as we described in Section 5.2. In this case, little can be done in this situation except to try to predict when, in the future, synchronization will be needed. However, using our model of local clocks and explicit conversions (Section 5.2.4), a superior solution falls out quite naturally: *post-facto synchronization.* That is, a node's local clock runs, unsynchronized, at its natural rate, and is used to timestamp events when they occur. After an event of interest, a synchronization process can be started, allowing timestamps from distributed nodes to be reconciled. Waste is reduced because the system avoids performing synchronization at times when it's not necessary. This scheme works much better than the predictive version: it is much easier to characterize the past than it is to predict the future.

In this chapter, we describe two forms of post-facto synchronization in more detail. The first is *single-pulse synchronization.* To reach its best accuracy, this method requires some advance calibration, but can reconstruct a precise timescale quite quickly when needed. Second, we describe the application of our Reference-Broadcast Synchronization scheme to the post-facto problem. In this scheme, the required convergence time is longer, but effective post-facto synchronization can be performed with no *a priori* knowledge.

8.1   Single-Pulse Synchronization

The simplest form of post-facto synchronization is *single-pulse synchronization.* That is, nodes' clocks are normally *un*synchronized. When an event occurs, each node records the time of the event with respect to its own local clock. Immediately afterward, a "third party" node—acting as a beacon—broadcasts a synchronization pulse to all nodes in the area using its radio. Nodes that

receive this pulse use it as an instantaneous time reference and can normalize their stimulus timestamps with respect to that reference.

This kind of synchronization is not applicable in all situations, of course: it is limited in scope to the transmit range of the beacon and creates only an "instant" of synchronized time. This makes it inappropriate for an application that needs to communicate a timestamp over long distances or times. However, it is simple, and can be useful if the required synchronization scope is entirely within the broadcast domain of a single node.

8.1.1 Expected Sources of Error

There are three main factors that affect the accuracy and precision achievable by single-pulse synchronization. Roughly in order of importance, they are the receiver clock skew, variable delays in the receivers, and propagation delay of the synchronization pulse.

- *Skew in the receivers' local clocks.* Our scheme requires that each receiver accurately measure the interval that elapses between their detection of the event and the arrival of the synchronization pulse. However, nodes' clocks do not run at exactly the same rate, causing error in that measurement. Since clock skew among the group will cause the achievable error bound to decay as time elapses between the stimulus and pulse, it is important to minimize this interval.

  One way of reducing this error is to use NTP to discipline the frequency of each node's oscillator. This exemplifies our idea of multi-modal synchronization. Although running NTP "full-time" defeats one of the original goals of keeping the main processor or radio off, it can still be useful for

119

frequency discipline (much more so than for phase correction) at very low duty cycles.

- *Variable delays on the receivers.* Even if the synchronization signal arrives at the same instant at all receivers, there is no guarantee that each receiver will *detect* the signal at the same instant. Nondeterminism in the detection hardware and operating system issues such as variable interrupt latency can contribute unpredictable delays that are inconsistent across receivers. The detection of the event itself (audio, seismic pulses, etc.) may also have nondeterministic delays associated with it. These delays will contribute directly to the synchronization error.

  As in RBS, single-pulse synchronization avoids error due to variable delays in the *sender* by considering the sender of the sync pulse to be a "third party." That is, the receivers are considered to be synchronized only with each other, not with the beacon.

  It is interesting to note that the error caused by variable delay is the same irrespective of the time elapsed between the event and the sync pulse. In contrast, error due to clock skew grows as time elapses between the event and sync pulse.

- *Propagation delay of the synchronization pulse.* Our method assumes that the synchronization pulse is an absolute time reference at the instant of its arrival—that is, that it arrives at every node at exactly the same time. In reality, this is not the case due to the finite propagation speed of RF signals. Synchronization will never be achievable with accuracy better than the difference in the propagation delay between the various receivers and the synchronization beacon.

This source of error makes our technique most useful when comparing arrival times of phenomena that propagate much more slowly than RF, such as audio. The six-order-of-magnitude difference in the speed of RF and audio has been similarly exploited in the past in systems such as the ORL's Active Bat [WJH97] and Girod's acoustic rangefinder [GE01].

8.1.2 Empirical Study

We designed an experiment to characterize the performance of our post-facto synchronization scheme. The experiment attempts to measure the sources of error described in the previous section by delivering a stimulus to each receiver at the same instant, and asking the receivers to timestamp the arrival time of that stimulus with respect to a synchronization pulse delivered via the same mechanism. Ideally, if there are no variable delays in the receivers or skew among the receivers' local oscillators, the times reported for the stimulus should be identical. In reality, these sources of error cause the dispersion among the reported times to grow as more time elapses between the stimulus and the sync pulse. The decay in the error bound should happen more slowly if NTP is simultaneously used to discipline the frequency of the receivers' oscillators.

We realized this experiment with one sender and ten receivers, each of which was ordinary PC hardware (Dell OptiPlex GX1 workstations) running the Red-Hat Linux operating system. Each stimulus and sync pulse was a simple TTL logic signal sent and received by the standard PC parallel port.[1] In each trial, each receiver reported its perceived elapsed time between the stimulus and synchronization pulse according to the system clock, which has 1μsec resolution.

---

[1]This was accomplished using the author's parallel port pin programming library for Linux, *parapin*, which is freely available at http://www.circlemud.org/jelson/software/parapin

We defined the dispersion to be the standard deviation from the mean of these reported values. To minimize the variable delay introduced by the operating system, the times of the incoming pulses were recorded by the parallel port interrupt handler using a Linux kernel module.

In order to understand how dispersion is affected by the time elapsed between stimulus and sync pulse, we tested the dispersion for 21 different values of this elapsed time, ranging from $2^4\mu$sec to $2^{24}\mu$sec (16μsec to 16.8 seconds). For each elapsed-time value, we performed 50 trials and reported the mean. These 1,050 trials were performed in a random order over the course of one hour to minimize the effects of systematic error (e.g. changes in network activity that affect interrupt latency).

We performed this entire experiment using three different configurations:

1. The experiment was run on the "raw clock": that is, while the receivers' clocks were not disciplined by any external frequency standard.

2. An NTPv3 client was started on each receiver and allowed to synchronize (via Ethernet) to our lab's stratum-1 GPS clock for ten days. The experiment was then repeated while NTP was running.

3. NTP's external time source was removed, and the NTP daemon was allowed to free-run for several days using its last-known estimates of the local clock's frequency. The experiment was then repeated.

To compare our post-facto method to the error bound achievable by NTP alone, we recorded two different stimulus-arrival timestamps when running the experiment in Configuration 2: the time with respect to the sync pulse *and* the time according to the NTP-disciplined local clock. Similar to the other configurations, a dispersion value for NTP was computed for each stimulus by computing
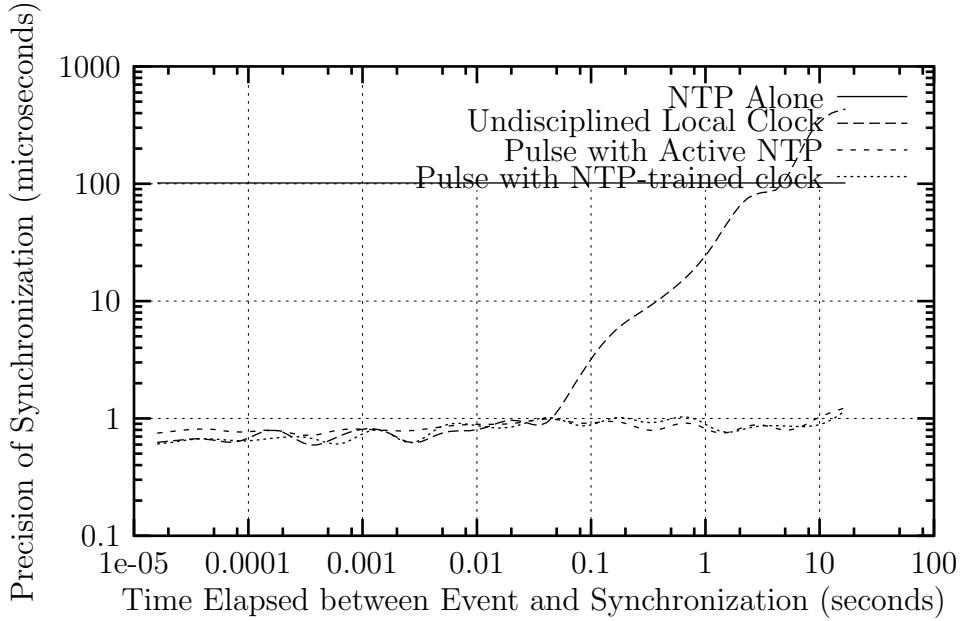
Figure 8.1: Synchronization error using post-facto time synchronization without external frequency discipline, with discipline from an active NTP time source, and with free-running NTP discipline (external time source removed after the oscillator's frequency was estimated). These are compared to the error bound achievable with NTP alone (the horizontal line near 100μsec). The breakpoint seen near 50msec is where error due to clock skew, which grows proportionally with the time elapsed from stimulus to sync pulse, overcomes other sources of error that are independent of this interval. Each point represents the dispersion experienced among 10 receivers, averaged over 50 trials.

the standard deviation from the mean of the reported timestamps. The horizontal line in Figure 8.1 is the mean of those 1,050 dispersion values—101.70 μsec.

Our results are shown in Figure 8.1. [2]

---

[2] "The joy of engineering is to find a straight line on a double logarithmic diagram." –Thomas Koenig

8.1.3 Discussion

The results shown in Figure 8.1 illuminate a number of aspects of the system. First, the experiment gives insight into the nature of its error sources. The results with NTP-disciplined clock case are equivalent to undisciplined clocks when the interval is less than $\approx$ 50msec, suggesting that the primary source of error in these cases is variable delays on the receiver (for example, due to interrupt latency or the sampling rate in the analog-to-digital conversion hardware in the PC parallel port). Beyond 50msec, the two experiments diverge, suggesting that clock skew becomes the primary source of error at this point.

Overall, the performance of post-facto synchronization was quite good. When NTP was used to discipline the local oscillator's frequency, an error bound very near to the clock's resolution of 1μsec was achieved. This is significantly better than the 100μsec achieved by NTP alone. Clearly, the combination of NTP's frequency estimation with the sync pulse's instantaneous phase correction was very effective. Indeed, the multi-modal combination's maximum error is better than either mode can achieve alone. We find this a very encouraging indicator for the multi-modal synchronization framework we proposed in Section 5.2.1.

Without NTP discipline, the post-facto method still performs reasonably well for short intervals between stimulus and sync pulse. For longer intervals, we are at the mercy of happenstance: the error depends on the natural frequencies of whatever oscillators we happen to have in our receiver set.

Perhaps the most exciting result, however, is shown in the experiment where NTP disciplined the nodes' local clocks using only the last-known-estimate of frequency, after the external time source was removed. The achievable error bound was 1μsec: the limit of our clock's resolution and, more importantly, exactly the same as the result with NTP and an active external time standard.

This result is important because it shows that extremely low-energy and low-error time synchronization is possible: after an initial frequency-training period, nodes might be able to keep their radios off for days and still instantly acquire a timebase within 1μsec when an event of interest arrives. That result is made possible by the multi-modal synchronization; the frequency correction provided by free-running NTP is not good enough to keep clocks precisely in phase over time due to accumulated error. (In the free-running NTP experiment, the accuracy of the timestamps when not normalized by the sync pulse was only in the tens of milliseconds.)

All of these results, while encouraging, do come with a number of caveats. Our experiments results were performed under idealized laboratory conditions, using (equal-length) cables to directly connect the sender to the receivers. Real world conditions will require wireless links that are likely far more complex with more opportunities for variable delay. In addition, the relatively constant ambient temperature reduced the oscillators' frequency drift over time. A real sensor network deployed outdoors might not be able let NTP free-run without an external time source for as long as we did in our experiment.

8.2   Post-Facto Synchronization with RBS

Successful single-pulse synchronization after long (> 50ms) intervals, as described in the previous section, really consists of two separate steps:

1. The relative clock skew of nodes is estimated *a priori.* Our example used NTP to perform this estimation, which can take several hours, as we saw in Figure 4.3.

2. After an event, a single pulse is used to correct phase, under the assumption

that the frequencies of the clocks are still the same as when computed in the first step.

There can, however, be situations where the *a priori* calibrations of frequencies can not be performed. For example, nodes that have never previously been in contact with each other may want to synchronize after an event they both witness. An interesting question to ask, then, is: how well can we do when the nodes start with no mutually shared information at all?

To test this scenario, we turned to our RBS implementation, as described in Chapter 6. An interesting facet of the RBS clock skew estimator pictured in Figure 6.4 is that it is is also an effective form of post-facto synchronization. After an event of interest, nodes can power their radios up and exchange sync pulses until the best-fit line that relates nodes' clocks to each other has been computed satisfactorily (e.g., the RMS error has fallen below some threshold). The best-fit line can then simply be extrapolate backward to estimate relative phase errors at times in the past—for example, at the time of the event.

Compared to the single-pulse, the RBS technique does have some disadvantages. Since many pulses (e.g., 30) are required instead of just one, the convergence time will be longer. The longer convergence time, in turn, implies higher phase error: by the time the experiment is complete, the nodes' relative frequencies might have drifted away from the value at the moment of the experiment.

8.2.1   Empirical Study

To test post-fact RBS, we used Motes as "network interfaces," rather than as stand-alone nodes as described in Section 6.3. Motes were connected via a 9600-baud serial link to PC104-based single-board-computers running the Linux oper-
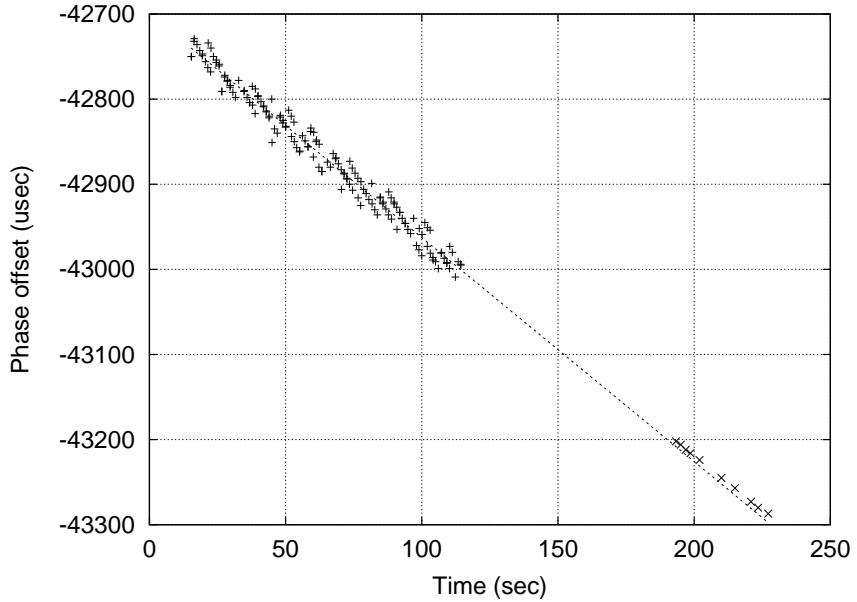
Figure 8.2: Post-facto synchronization using RBS to estimate both skew and phase. RBS synchronized clocks on PC104-compatible single-board-computers using a Mote as NIC. The points near $x = 200$ are reference pulses, which show a synchronization error of 7.4μsec 60 seconds after the last sync pulse.

ating system. In some sense, this makes the problem more difficult because we are extending the path between the receiver (the Mote) and the clock to be synchronized (under Linux). We modified the Linux kernel to timestamp serial-port interrupts, allowing us to accurately associate complete over-the-air messages with kernel timestamps.

One of these "Mote-NICs" sent reference broadcasts to two receivers for two minutes. Each reception was timestamped by the Linux kernel. After one minute of silence, we injected 10 reference pulses into the PC's parallel port; these pulses appear at the right of the figure. We computed the best-fit line based on the Mote-NIC pulses, after automatic outlier rejection. There was a 7.4μsec residual error between the estimate and the reference pulses. The experiment is shown in Figure 8.2.

8.2.2 Discussion

To date, our simple experimentation with RBS for post-facto synchronization has not been comprehensive. However, we believe the results are very encouraging: even 60 seconds after an event, RBS can reconcile clocks to a phase error that is below the mean jitter of the detector. (Recall from Section 6.3 that the Mote's RMS bit detection jitter was 11.2 μsec; in our experiment, phase error was 7.4μsec.) Unlike single-pulse synchronization, RBS can achieve this result when there is no *a priori* shared knowledge or calibration between the nodes in question.

Of course, the best-fit line computed by RBS can be extrapolated either backward or forward in time. That is, in addition to post-facto synchronization, this technique is equally applicable to coordinated actuation of events in the future. This can be important in sensor networks; we discussed a number of uses for coordinated actuation in Section 3.3.

Finally, it is interesting to note the role played by our new "explicit conversions" service model for time synchronization, proposed in Section 5.2.4. Under this model, the normal way applications interface to the time synchronization system is to first retrieve a local timestamp during events of interest, then ask for explicit conversions to other time-bases. Using this model, post-facto synchronization has the same interface as normal synchronization, except perhaps that the conversion step does not succeed until some time significantly later than the timestamping step. Synchronization services can also provide an API to allow applications to indicate that they need synchronization—i.e., "start sending sync pulses now."

8.3 Summary

In this chapter, we explored two different forms of post-facto synchronization. The first, *single-pulse synchronization*, requires frequency calibration at the time the network is deployed (and perhaps periodically afterward). With this information, high-precision retrospective timescales can be quickly constructed using a single packet. Second, we have shown that Reference-Broadcast Synchronization can be used for post-facto synchronization. In our test, the phase error of a retrospective RBS timescale was still less than the bit-detection jitter when 60 seconds elapsed between the last synchronization packet and first timestamped event.

Post-facto synchronization can be a highly effective way to save energy in a sensor network. In scenarios where synchronization is needed only occasionally and unpredictably, traditional algorithms waste energy by keeping the clock synchronized at all times, just in case it is needed. In contrast, post-facto synchronization allows the clock to stay *un*-disciplined. Timestamps from free-running clocks are reconciled after a node identifies an event of interest—specifically, one that justifies the energy cost of performing synchronization.

CHAPTER 9

Application Experience

Be wise in the use of time. The question in life is not how much time
do we have? The question is what shall we do with it.

—Anna Robertson Brown

To date, our RBS implementation has been applied, both within and without
our research group, in three distributed sensor network applications requiring
high-precision time synchronization. Each uses RBS-synchronized clocks to pre-
cisely measure the arrival time of acoustic signals. Most audio codecs sample
the channel at 48KHz, or $\approx 21\mu$sec per sample. As we have seen in previous sec-
tions, the precision nominally achieved by RBS is significantly better. Distributed
acoustic sensors have thus been able to correlate their acoustic time-series down
to individual samples. This has been useful in a number of contexts.

9.1 Automatic Mote Localization System

Joint work with Lewis Girod and Vladimir Bychkovskiy

Spatial localization is a crucial building block for many sensor network ap-
plications. Indeed, virtually all the example applications we described in Chap-
ter 3—multi-sensor integration, array processing, coordinated actuation, and so
forth—require a common reference frame in both time *and* space. In the small-

est networks, it is possible to implement spatial localization of nodes manually, by carefully measuring and "hard-coding" the locations of the sensors. However, this is approach does not scale, is ill-suited to ad-hoc deployment, and does not work in networks with mobile nodes. Consequently, many researchers have focused attention on building systems that discover location automatically [WHF92, BHE03, WC02].

In this section, we discuss the development of a system, based on commercial off-the-shelf (COTS) components, which is capable of automatic localization and time synchronization with sufficient precision (on the order of 10cm and 10μsec) to support distributed, coherent signal processing. Our system's time synchronization is an implementation of Reference-Broadcast Synchronization (RBS), described in Chapter 6. Localization is based on an underlying ranging system that works by timing the flight of a wideband acoustic pulse, described in [GE01].

The remainder of this section is organized as follows. In Section 9.1.1, we describe the hardware platforms that compose our testbed. We give an overview of the software components of our system in Section 9.1.2. A more detailed description of the subsystems is found in Section 9.1.3 (time synchronization) and Section 9.1.4 (acoustic ranging and localization).

9.1.1 Hardware Platforms

Although Moore's law predicts that hardware for sensor networks will inexorably become smaller, cheaper, and more powerful, technological advances will never prevent the need to make tradeoffs. Even as our notions of metrics such as "fast" and "small" evolve, there will always be compromises: nodes will need to be faster *or* more energy-efficient, smaller *or* more capable, cheaper *or* more durable.

Instead of choosing a single hardware platform that makes a particular set

of compromises, we believe an effective design is one that uses a tiered platform consisting of a heterogeneous collection of hardware. Small, cheap, and computationally limited nodes ("motes", after the Berkeley Smart Dust project [KKP99]) can be used more effectively by augmenting the network with larger, faster, and more expensive hardware ("cluster heads"). An analogy can be made to the memory hierarchy commonly found in desktop computer systems. CPUs typically have an expensive but fast on-chip cache, backed by slower but larger L2 cache, main memory, and ultimately on-disk swap space. This organization, combined with a tendency in computation for locality of reference, results in a memory system that appears to be as large and as cheap (per-byte) as the swap space, but as fast as the on-chip cache memory. In sensor networks, where localized algorithms are a primary design goal [IGE00], similar benefits can be realized by creating the network from a spectrum of hardware ranging from small, cheap, and numerous, to large, expensive, and powerful.

9.1.1.1 Motes

The smallest nodes in our testbed are the "COTS Mote," originally developed at U.C. Berkeley [KKP99, HSW00] and now commercially available from Crossbow Technologies[1]. The Mica 1 model is equipped with a low-power, 4MHz, 16-bit microprocessor (Atmel 90LS8535), with 8K of program memory and 512 bytes of SRAM. An integrated RF Monolithics 916.50 MHz transceiver (TR1000) provides narrowband wireless connectivity at 19.2Kbps. Communication with a directly attached host is possible via a serial port. A number of GPIO pins, A/D, and D/A converters are also available for I/O purposes.

Different versions of the basic design have various combinations of sensors:

---

[1]http://www.xbow.com
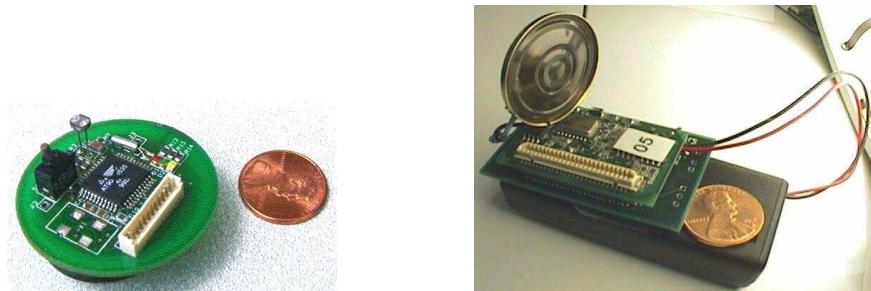
Figure 9.1: *left*) Pister's "COTS Mote", developed at U.C. Berkeley. *right*) The "Acoustic Mote" used to support acoustic ranging, developed by Naim Busek at the UCLA LECS laboratory.

temperature, light, humidity, acceleration, and so forth. In our localization testbed, we use a locally developed variation of the mote, the "acoustic mote," on which one of the D/A converter outputs is attached to an audio amplifier circuit and speaker. The original COTS Mote and our locally developed acoustic mote are pictured in Figure 9.1.

The motes' operating environment is TinyOS [HSW00], a small, event-based operating system developed specifically for the mote by Hill, et.al. at U.C. Berkeley.

9.1.1.2 Cluster Heads

Although the motes have advantages (small, cheap, low-power, long-lived), they have very limited capabilities. For example, it is impossible to do anything more than the most basic forms of signal processing: the mote is too slow to process a time series in real-time, and doesn't have enough memory to buffer more than a few dozen samples for offline analysis. We have found that it is possible to implement much richer applications if there are a few larger, more computationally endowed nodes available in the network as well.

Our cluster heads are Compaq iPAQ 3760s, which are handheld, battery-

powered devices normally meant to be used as PDAs. We selected the iPAQ because it has reasonable battery life, has the peripherals needed by our project, supports Linux, is readily available, and is usable right off the shelf.[2] The iPAQ has a 206 MHz Intel StrongARM-1110 processor, 64MB of RAM and 32MB of FLASH for persistent storage. The standard model also comes equipped with a built-in speaker and microphone which we use for acoustic ranging, as we will see in Section 9.1.4. Finally, the iPAQs have a serial port, and a PCMCIA bus, for which a wide variety of peripherals are available. All of our iPAQs have spread-spectrum wireless Ethernet cards (802.11b direct sequence, 11Mbit/sec), making them capable of communicating at higher bandwidth, longer range, and with greater reliability than the motes.

Our testbed's iPAQs use the StrongARM port of the Linux operating system (the "Familiar" distribution [FAM]). This combination of hardware and operating system provides a powerful and convenient development environment similar to a standard desktop operating system.

9.1.2 System Overview

Ultimately, the goal of our system is to first establish a coordinate system defined by the positions of the iPAQs at startup, then continuously monitor the position of the motes within this coordinate system. We assume the iPAQs are initially placed around a room "at random" but then remain in fixed locations, or move rarely. The iPAQs form an ad-hoc infrastructure, while the smaller motes are assumed to be constantly in motion—perhaps attached to people or objects that are being tracked as they move through the environment.

---

[2]Previous incarnations of our testbed [CEE01] were based on hardware platforms that we integrated ourselves from components; we have since learned not to underestimate the value of a platform that comes with its own integrated power supply, enclosure, I/O, etc.

The functional components of the localization system are shown in Figure 9.2. Localization is based on an acoustic ranging system that uses a wideband pseudo-noise sequence to measure the time of flight of sound from one point to another (*bottom panel*). Ranging is first used to determine iPAQ-to-iPAQ distances and federate an iPAQ coordinate system; this mode uses both the iPAQ's speaker and microphone (*center panel*). Once the coordinate system has been established, ranging from an acoustic mote's emitter to the iPAQs' microphones is used to localize motes within this coordinate system (*top panel*).

Many existing acoustic ranging systems use a somewhat ad-hoc time synchronization strategy that assumes there is a tight coupling between the acoustic and RF components in the system. Those systems are designed to generate an acoustic pulse and an RF synchronization packet at exactly the same time. We have found this to be a poor design choice for a number of reasons—for example, the system tends to be more complex due to tight inter-module dependencies, and ranging experiments are vulnerable to individual lost packets. In contrast, our system contains a separate module that continuously maintains timebase conversion metrics among all the components in the system, providing this information to the ranging system when necessary. By abstracting this part of the system away, the overall design is simplified and the system becomes much more robust to packet loss. In addition, the precision of the synchronization is improved: by averaging over many packet observations, outliers can be rejected and clock skew corrections are possible.

We mentioned in the previous section that motes use narrowband 900MHz radios to communicate with each other, while the iPAQs use 802.11b wireless Ethernet cards. To bridge the gap between these two domains, several of the iPAQs are configured with a "MoteNIC"—that is, a mote attached to the iPAQ

Localization Service
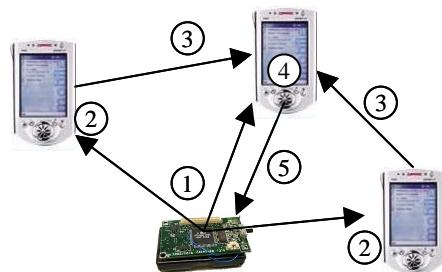
1. Mote emits acoustic ranging signal
2. Receiver process detects signal
3. Local coordination with nearby receivers
4. Position estimated within local system
5. Coordinates published back to mote

Infrastructure Configuration Algorithms
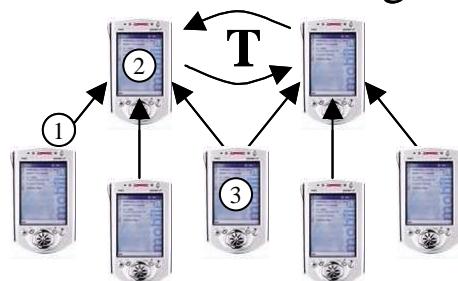
1. Results published to cluster head(s)
2. Cluster head runs minimization to compute cluster coordinate system
3. Dual-homed nodes compute coordinate transforms between neighboring coordinate systems

Ranging System

1. Pairwise time synchronization (RBS)
2. Wideband acoustic detector
3. Statistical filtering of ranging results
4. Publish results to cluster head

RBS Sync

Acoustic Pulse

Time
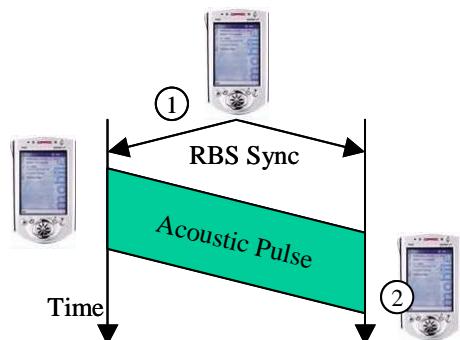
Figure 9.2: High-level System Diagram. At the lowest layer, a ranging system generates pairwise range values. These ranges are propagated to a cluster head that runs an optimization algorithm to construct a consistent coordinate system and reject the many forms of error in the source data. Once a coordinate system has been computed, the system can localize devices in the region.

via the serial port. Such motes can act as a network interface visible to processes running on the iPAQ (under Linux). These iPAQs are then capable of routing information between the mote and iPAQ domains.

9.1.3 Time Synchronization

The time synchronization module is responsible for computing parameters that relate the phase and frequency of all of the system's clocks to one another. This is, perhaps, a more complex task than it might seem—in addition to being a distributed system, it is a heterogeneous one. Specifically, there are three types of components in the system whose clocks run independently and must be reconciled:

- The system clock in each mote

- The system clock in each iPAQ

- Each iPAQ's codec sample clock

The synchronization service builds up a table of conversion parameters that relate many of the different clocks in the system to each other. A series of conversions may be necessary to convert a timestamp in one timebase to a timestamp in another. For example, consider the configuration shown in Figure 9.3. iPAQ 1 is configured as a gateway node, i.e., with a MoteNIC as described in Section 9.1.2. Imagine that we wish to measure the time of flight of a sound from an acoustic mote to iPAQ 2. This means we need to relate the timebase of the acoustic mote to the timebase of iPAQ 2's codec sample clock. This requires a series of 4 conversions:

1. Network time synchronization (via mote radios) from the acoustic mote's

802.11 Broadcast Channel

IPAQ 1 IPAQ 2 IPAQ 3

Serial

Mote Broadcast Channel

MoteNIC Acoustic Mote Acoustic Mote
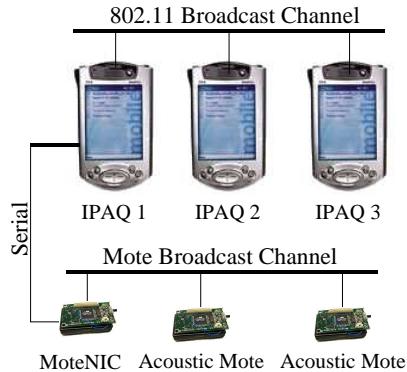
Figure 9.3: Our hardware configuration, where multi-hop timesync is required to relate the acoustic mote's timebase to the iPAQs.

clock to mote clock in iPAQ 1's MoteNIC

2. Intra-node synchronization from iPAQ 1's mote clock to iPAQ 1's system clock

3. Network time synchronization from iPAQ 1 to iPAQ 2 (via 802.11)

4. Intra-node synchronization from iPAQ 2's system clock to iPAQ 2's codec sample clock

The methods for intra-node and network time synchronization will be discussed in detail below. On each iPAQ, those synchronization methods populate a table of conversion parameters between the clocks in the system, annotated with an estimate of the RMS error of each conversion.

Figure 9.4 shows a snapshot of our GUI visualizing a running time synchronization system. Given the parameters that make up this graph, the series of parameters that make up the minimum-error conversion between the desired source and destination timescales is automatically computed, using a weighted shortest-path search algorithm.

138
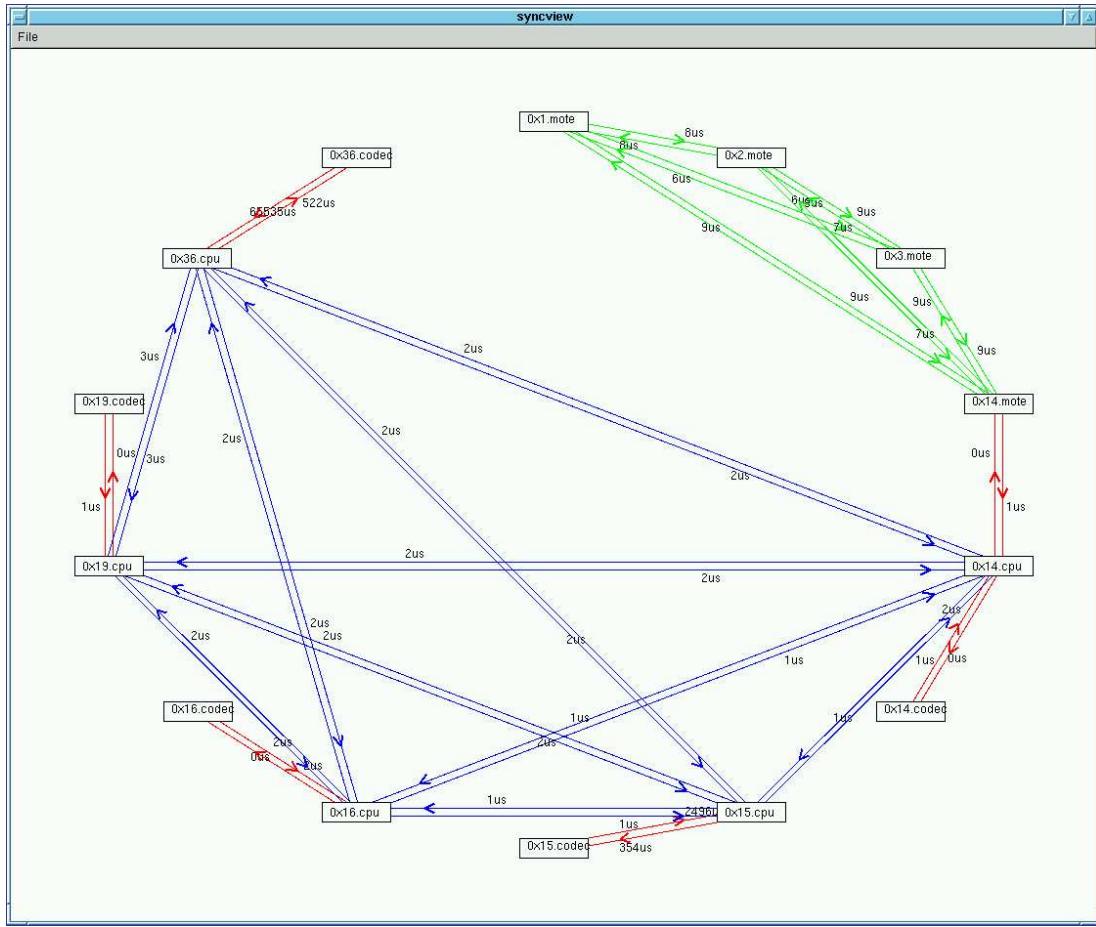
Figure 9.4: "Syncview", our GUI that visualizes the state of the time synchronization system. All time-sync relationships in the system are shown. Intra-node synchronization forms relationships from an iPAQ's CPU to its audio codec, and from an iPAQ CPU to a directly attached MoteNIC (if any). RBS is used to form relationships from Mote to Mote, and from iPAQ to iPAQ CPU via 802.11.

9.1.3.1 Network time synchronization

In our system, network time synchronization (mote-to-mote and iPAQ-to-iPAQ) is performed using an implementation of Reference-Broadcast Synchronization as we described in Chapters 6 and 7. Our RBS daemon simultaneously acts in both "sender" and "receiver" roles. Every 10 seconds (slightly randomized to avoid unintended synchronization), each daemon emits a pulse packet with a sequence number and sender ID. The daemon also watches for such packets to arrive; it timestamps them and periodically sends a report of these timestamps back to the pulse sender along with its receiver ID. The pulse sender collects all of the pulse reception reports and computes clock conversion parameters between each pair of nodes that heard its broadcasts. These parameters are then broadcast back to local neighbors. The RBS daemons that receive these parameters make them available to users. (RBS never sets the nodes' clocks, but rather provides a user library that converts UNIX *timeval*s from one node ID to another.)

As described in Section 6.2.2, our RBS implementation performs a least-squares linear regression on the time series of phase differences between nodes, after automatic outlier rejection. This offers a fast, closed-form method for finding the best fit line through the phase error observations over time. The frequency and phase of the local node's clock with respect to the remote node can be recovered from the slope and intercept of the line. An example of such a linear regression on real data from our system is shown Figure 9.5. Each regression is based on a window of the 30 most recent pulse reception reports.

Motes and iPAQs both participate in this scheme separately. That is, iPAQs synchronize with each other using 802.11 broadcasts, while motes synchronize with each other using their narrowband radio broadcasts. The basic scheme is the same on both motes and iPAQs, except that motes do not compute their
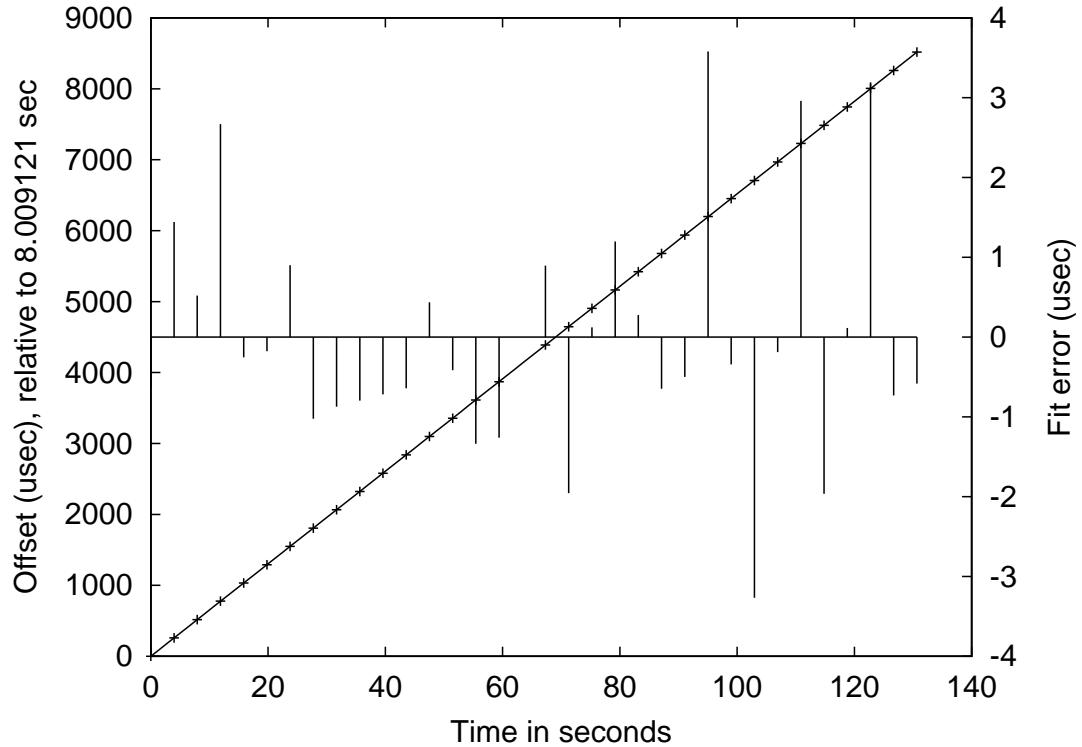
Figure 9.5: Typical linear regression to recover relative phase and skew between two nodes in the system using RBS. This picture shows synchronization between two Compaq iPAQs via 11Mbit/sec 802.11 wireless Ethernet. Each point represents the phase offset between two nodes as implied by the value of their clocks after receiving a reference broadcast. A node can compute a least-squared-error fit to these observations (*diagonal line*), and thus convert time values between its own clock and that of its peer. The vertical impulses (read with respect to the $y2$ axis) show the distance of each point from the best-fit line.

own conversion parameters, as iPAQs do—they forward all of their broadcast reception reports to an iPAQ for post-processing instead.

In practice, iPAQ-to-iPAQ synchronization via 802.11 has an error of $\approx$ 1.5μsec; this is limited primarily by the iPAQ's 1μsec clock resolution. Motes use slower (19.2kbit/sec) radios with a bit time of 53μsec; nevertheless, RBS achieves 10μsec error through averaging and outlier rejection.

Of course, motes and iPAQs can not directly observe each other's broadcasts since they use different radios (different frequencies, different MAC layers, etc.). An extra piece is needed to relate the mote timescales with the iPAQ timescales. This is where intra-node synchronization comes into play.

9.1.3.2 Intra-node Synchronization

As we discussed at the beginning of this section, each iPAQ can consist of three separate clocks: 1) the system clock; 2) the audio codec's sample clock, and 3) if the iPAQ has a MoteNIC attached via serial port, the mote's system clock. In addition to supporting RBS as described above, our time service daemon also supports synchronization between components within a system. The system's device drivers periodically inject pairs of time values into the time daemon, where each pair represents the value of two of the system's clocks at the same instant. This allows the clocks to be related. There are two types of intra-node synchronization in our system:

- *iPAQ to MoteNIC*. One of the output pins of the serial port is attached to an interrupt-generating input on the MoteNIC. Periodically, the iPAQ raises its output high and records the time according to its CPU clock. The MoteNIC timestamps the resulting interrupts according to its internal

clock, and reports the timestamps back to the iPAQ via the serial link. Once the MoteNIC device driver receives this interrupt time back from the MoteNIC, it injects the pair of time values into the time daemon. Each of these points represents a single mapping of the iPAQ's system clock timescale to the MoteNIC's system clock timescale.

- *iPAQ to codec sample clock.* Cheap consumer-grade audio codecs such as the variety typically found in a low-end PC or handheld device tend to have non-deterministic latency bounds when they are asked to start recording or playback. It is important to minimize these effects—delay between the time we ask the codec to start sampling and the time it actually starts contribute directly to errors in the time-of-flight measurement. We have found that this problem can be avoided by running the codec continuously, timestamping each block of audio data as it arrives. Our system includes an "audio server" that continuously records, buffering the most recent 10 seconds of input and making it available to the acoustic ranging process. With the help of a modification to the Linux kernel's codec device driver, the audio server also timestamps each DMA transfer from the codec's chipset as it arrives. It then injects synchronization pairs into the time daemon, each consisting of an audio sample number and corresponding system clock time when the DMA transfer completed.

As these time pairs are fed to the time synchronization daemon, it computes conversion parameters that allow the iPAQ CPU clock to be related to the MoteNIC and codec clocks. The daemon uses the same linear least-squares regression and outlier rejection as it does for RBS. This makes it very robust against outliers due to (for example) an occasional late DMA transfer or late interrupt on the mote.

These intra-node parameters allow us to complete the synchronization chain: from acoustic mote to MoteNIC (via RBS); MoteNIC to attached iPAQ 1 (via pairs synchronization); iPAQ 1 to iPAQ 2 (via RBS over 802.11), and finally iPAQ 2 to its codec sample clock (via pairs synchronization). The exact synchronization path is computed automatically, transparently to users of the time sync service. This significantly simplifies the localization service, as we will describe in the next section.

9.1.4 Localization

The localization subsystem consists of two main components. First, the acoustic ranging component estimates the distances between nodes in the network. Next, a coordinate system is constructed using the range estimates. In this section, we will describe the acoustic ranging system in detail. The details of coordinate system construction can be found in [GBE02].

Our acoustic ranging system, based on Girod's idea described in [GE01], uses a wideband pseudo-noise sequence to measure the time of flight of sound from one point to another. In the current implementation, the detector is implemented in software, and requires considerable memory and CPU time (far beyond the capabilities of a Mote). However, the high process gain of the detector enables long ranges, accurate phase determination, and excellent noise and multipath immunity. The use of wideband coded signals has two positive effects. The first advantage is that the frequency diversity of the wideband ranging signal enables it to work well in a variety of environments. Because we use audible sound, the wavelengths of sound making up the signal spans from a meter to a centimeter, increasing the resilience of the signal to scattering. The second advantage is that different emitters can select orthogonal codes that can be detected even when

they collide. This enables a drastic reduction in the system complexity, as it reduces the need for tight coordination and synchronization.

Because of the limited capabilities of the Mote as an emitter, we emit a position-modulated pulse train by toggling a digital output pin. Passed through the filter of a speaker, each pulse results in a decaying oscillation. The pseudo-noise code is represented by variations in the inter-pulse spacing, with an overall rate that is low enough to allow most of the oscillations to decay during each gap. The advantage of this scheme is that, even without exercising control over the dynamics of the speaker, precise timing on the Mote can reproduce the pulse timing with high precision, yielding a high degree of phase accuracy.

The bottom panel of Figure 9.2 shows how the ranging system works. Time-base conversion metrics between the sender and receiver are maintained by the synchronization service we described in Section 9.1.3. To compute the range between two devices, one device sends a message to the other, advertising that it is planning to emit an acoustic signal with a given code at a particular time, referenced in terms of its local timebase. Using the conversion metrics, the receiver can know approximately when to start listening. The audio DSP's are then sampled, and the resulting time series is compared with a locally generated reference signal using a sliding correlator.

In a sliding correlator, the correlation of two signals is computed at different relative phase offsets. Figure 9.7 shows a graph of correlation as a function of "lag." By analyzing this correlation function, we can estimate the most probable time of arrival of the ranging signal. The maximum value of the correlation function indicates the time of arrival of the strongest component of the signal. Figure 9.6 shows a portion of the observed signal, aligned with the reference signal at the "best" offset.
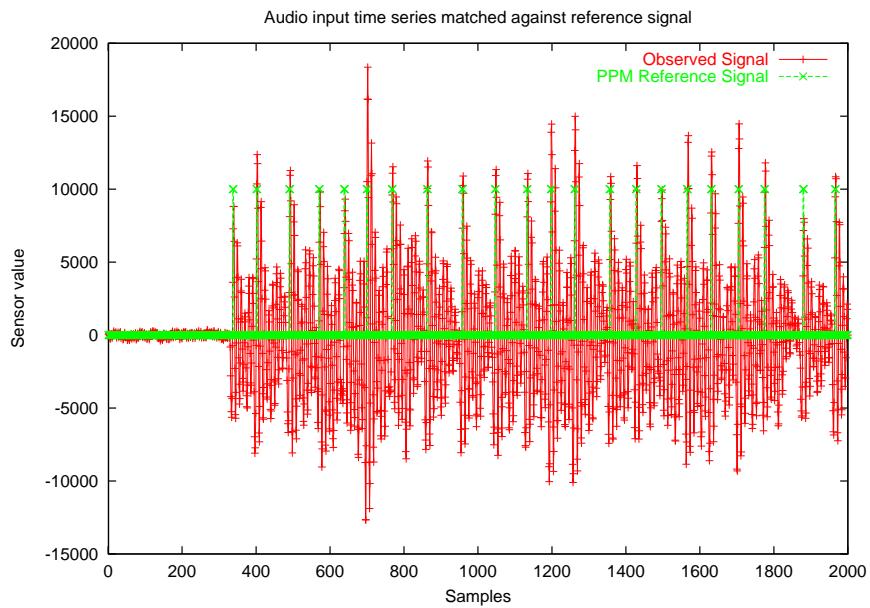
Figure 9.6: Pulse Position Modulated reference signal aligned with observed signal, captured in very low noise conditions. *Figure courtesy of Lewis Girod, UCLA LECS Laboratory*
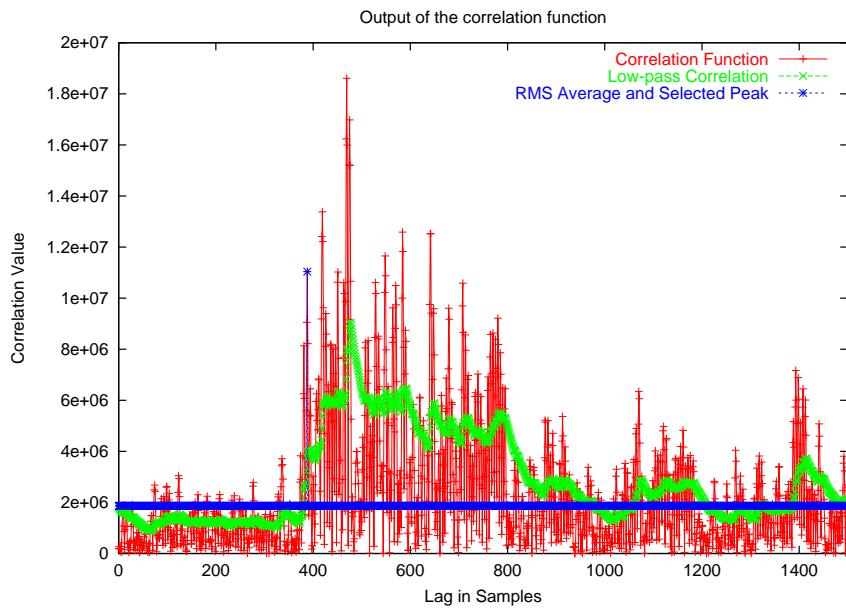
Figure 9.7: Correlation function and peak selection algorithm in multipath environment. *Figure courtesy of Lewis Girod, UCLA LECS Laboratory*

Echoes and other environmental distortions result in many successive peaks, among which the earliest peak represents the most direct path. The directional nature of speakers and microphones means that the maximum value of the correlation function often represents a strong reflection; for example, if the speaker is pointed at the ceiling. To resolve this problem, we use an adaptive noise threshold function based on a low-pass filter of the correlation function. We use this low-pass filter to select the earliest "cluster" of peaks, by finding the first region in which the low-pass filter crosses a fixed threshold of 1.5 times the RMS average of the correlation function. Within that region, we select the first peak after the low-pass filter crosses the RMS average. This algorithm tends to locate the first peaks from the region in which the power level abruptly increases.

Using the timebase conversion metrics and the offset of the earliest peak in the correlation function, the time of flight can be calculated. Given a model for the speed of sound in air, the time of flight can be converted to a distance. Since the speed of sound varies based on temperature and humidity, a value from a temperature sensor may be integrated into the model to get a more accurate distance estimate.

Once distance estimates are collected, multiple trials are combined statistically into a mean and variance, with outlier rejection. However, as we will see below, in general outliers cannot be completely filtered at this layer: long ranges resulting from detection of reflections in obstructed conditions typically exhibit a low variance.

We tested our prototype implementation by allowing iPAQ to form their initial coordinate system, then using it to measure the positions of motes in several locations. The results of these measurements are shown in Figure 9.9. The RMS position error in this set of nodes is 9.2cm.
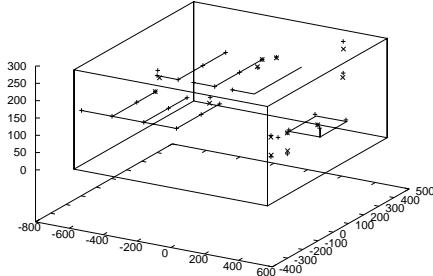
Figure 9.8: Positions of receivers after the configuration step. In this diagram only the receivers and four additional points are considered.
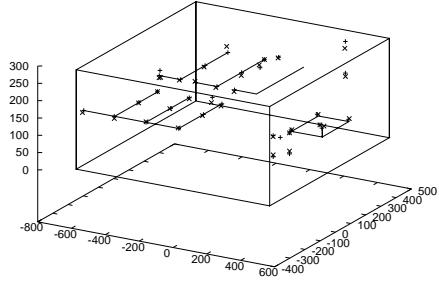
Figure 9.9: Positions of receivers and other observed devices. In both diagrams, 'X' indicates computed positions and '+' indicates ground truth. The lines represent cubicle walls and a table in the room. *Figures courtesy of Lewis Girod, UCLA LECS Laboratory*

In order to evaluate the performance of our localization system, we compared the results produced by our system to our ground truth measurements. For each position that our system estimated, we computed the distance between the computed value and ground truth. A histogram of these values is shown in Figure 9.10. This histogram plots position error on the X axis, and shows the percentage of measurements exhibiting less than that amount of error on the Y axis.

9.2 The DARPA SHM Program

Joint work with William Merrill, Lewis Girod, William Kaiser, Fredric Newberg, and Katayoun (Kathy) Sohrabi

Merrill *et al.* describe Sensoria Corporation's use of RBS in their implementation of a distributed, wireless embedded system capable of autonomous position
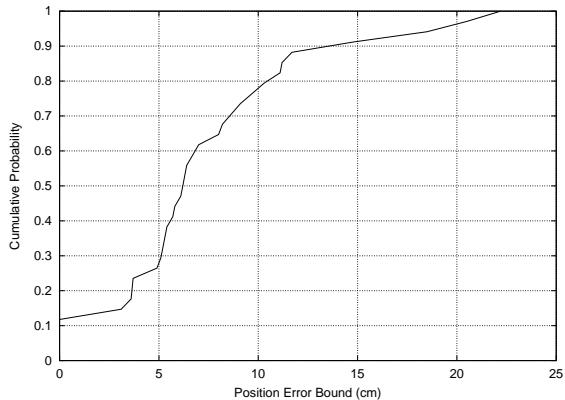
Figure 9.10: CDF of position error for the observations in our computed coordinate system. With 80% probability, the position error is less than 10cm.

location with accuracy on the order of 10cm [MGE02, SME02]. This system was built under the auspices of the DARPA SHM program [SHM] and has been field tested three times at Fort Leonard Wood, Missouri, in configurations of up to 75 nodes. The goal of the SHM application is to create a network of sensor nodes (pictured in Figure 9.11) that automatically maintain a minimum deployment density. That is, the nodes collaboratively build a geographic map of their relative positions, and physically rearrange themselves using solid-propellant rocket thrusters if a breach is detected.

SHM nodes are autonomous and self-configuring. At boot-up, or if network connectivity is lost, they perform automatic, distributed network construction: neighbor discovery, TDMA cluster formation, and leader election, as described in [SME02]. The neighbor discovery algorithm uses broadcast packets, so are also used as observable reference-broadcast events for RBS time synchronization. As in our mote localization system, nodes that observe broadcasts send their observations back to the broadcaster. The cluster-head computes relative phase and skew among the members of its cluster as described in Chapter 6; a typical fit is shown in Figure 9.12.

Figure 9.11: An SHM node built by the SAIC-led SHM team, including Sensoria, Ensign-Bickford, and Universal Propulsion. The node is 12cm in diameter. The top and bottom each have four solid-propellant rocket thrusters capable of moving the node approximately 8 meters in one of the four cardinal directions. The four holes on the top of the unit are the thruster nozzles. *Photo courtesy of Sensoria Corporation.*
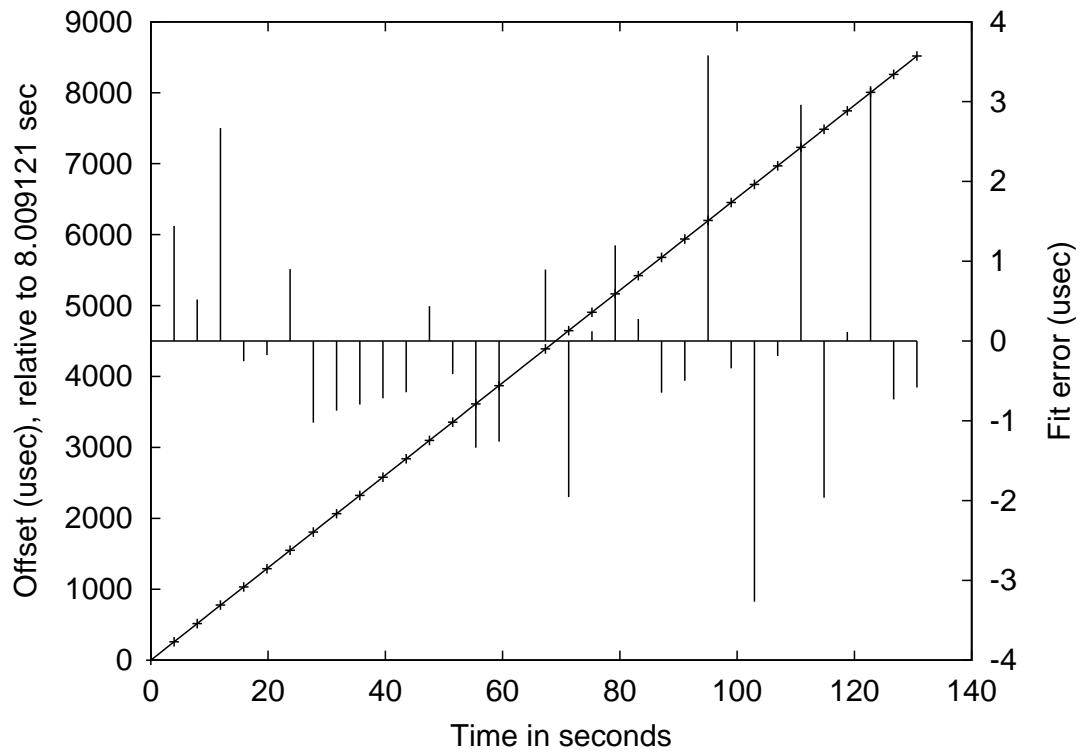
Figure 9.12: Typical RBS output using SHM's WIT2410 radios. Sending packets has high jitter (10s of milliseconds) because the radio has an RS-232 interface and uses a TDMA MAC layer. However, as seen in the figure, the reception process is very deterministic: most points fit within 3 microseconds.

The RBS daemon makes the conversion parameters available to the routing algorithm and packet forwarding mechanisms. Timestamps contained in data packets are time-converted hop-by-hop as they are forwarded through the network. The physical layer of the SHM radios forces each cluster to be a "star" topology; this guarantees that nodes serving as cluster-heads both have the RBS conversion parameters and are on the route of any packet leaving the cluster.

After RBS time synchronization is established, nodes begin an acoustic ranging and multilateration process, described in [MGE02]. The ranging process is similar to our Mote Localization system described in Section 9.1, but the multilateration process is a closed form computation that does not use a mass-spring model. The nodes collaboratively build a map of their relative positions after several minutes. Each node periodically checks that its neighbors are still alive. If a node is disabled or destroyed, those that remain detect the absence, and the network makes a "healing decision"—that is, one or more nodes move to fill in the open space.

The mobility subsystem consists of solid-propellant rocket thrusters on the top and bottom of the node that can be fired at the command of the host processor. Each charge produces a 100ms impulse, generating thrust that launches the node in a parabolic arc such that it comes to rest about 8 meters from its starting position. The thruster nozzles are oriented so that each charge can move the node in one of the four cardinal directions. Each node also has an internal compass and up/down sensor (accelerometer) so that subsequent healing decisions can take into account the final resting position of the node after its previous motion.

Although the network requires no control input, it does have a graphical interface that allows interactive debugging and inspection of system state. An example of this interface is shown in Figure 9.13.
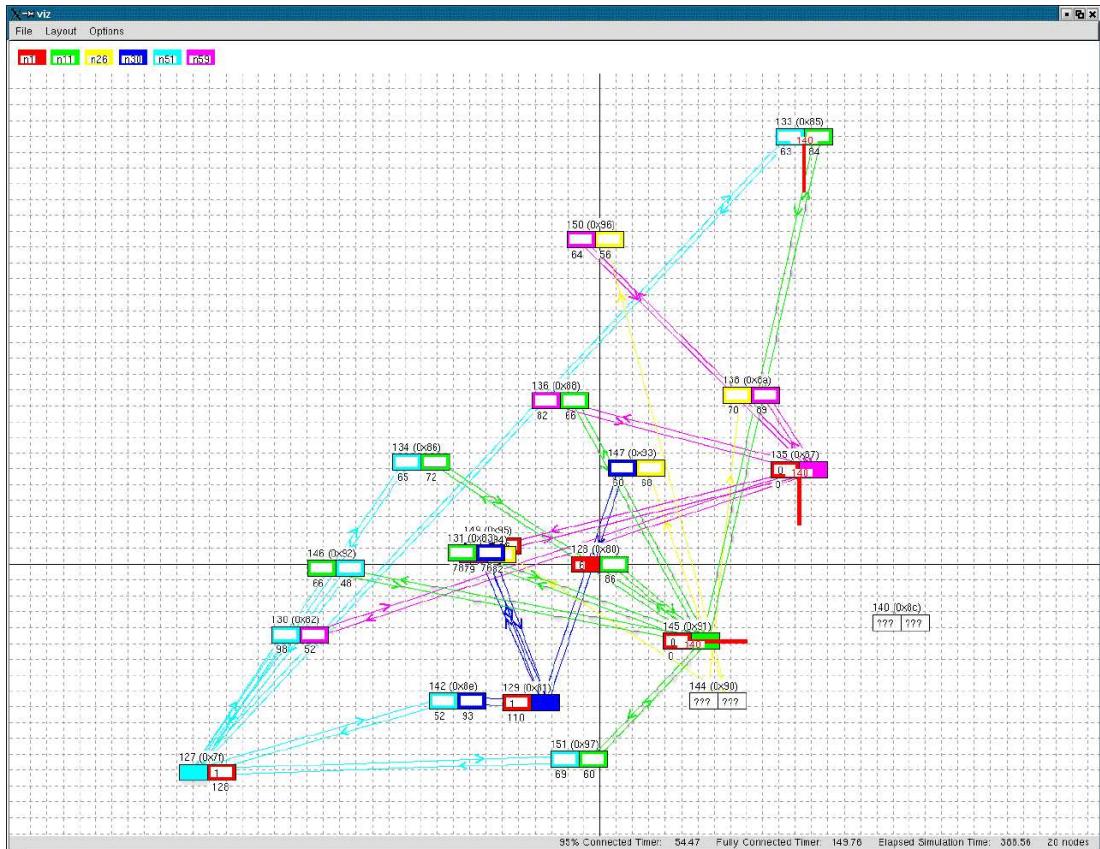
Figure 9.13: "Viz," the SHM GUI, shows information including the nodes' collaboratively computed geographic map, and their healing decisions. In this example, nodes 140 and 144 (lower right, denoted with "???") have been turned off to simulate destruction by an enemy. Nodes 133, 135 and 145 have displayed their healing decisions: they plan to move south, south and east, respectively. If the dead nodes are not revived within 30 seconds, the others will execute their healing decisions by firing rockets.

153

The SHM application is notable for a number of reasons. First, it represents a third hardware and radio platform on which RBS has been successfully implemented. Sensoria's "WINS NG" node is based around the Hitachi SH4 processor running Linux 2.4. Each node has two low-power 56Kbit/sec hybrid TDMA/FHSS radios with an RS-232 serial interface to the host processor. As with our 802.11 implementation, the firmware of this radio was opaque, making schemes that rely on tricks at the MAC layer impossible. The success of RBS here lends more weight to our proposition that RBS is not simply a peculiarity of one platform, but is likely to work across a wide variety of network hardware and MAC layers.

SHM is also noteworthy because it is a complete, fully realized sensor network application. RBS is central to its success, but it also relies on many other crucial services (e.g., soft-state neighbor discovery, automatic topology configuration, ad-hoc routing, reliable groupcast, acoustic ranging, distributed multilateration, collaborative healing), sensors (microphone, compass, accelerometer), and actuators (rockets, speakers). Many of our research systems demonstrate solutions to one aspect of problem, but SHM is a complete implementation of a real application. Its success was dramatic, and the system is now being considered by various branches of the armed forces for transition from a DARPA research program into a deployable system.

9.3   Blind Beamforming

Joint work with Hanbiao Wang, Len Yip, Lewis Girod, Daniela Maniezzo, Joe C. Chen, Ralph E. Hudson, and Kung Yao

Finally, we will briefly describe a third application of RBS: distributed blind

beamforming. Blind beamforming on acoustic signals has been studied by Yao *et al.* for a number of years [YHR98]. Their prototype arrays localize the source of sounds by computing phase differences of an acoustic signature received by sensors at different locations. This technique is similar to acoustic ranging, but does not require time-synchronization between the acoustic emitter and receiver. Instead of measuring the absolute value of the propagation delay, the array measures time *difference* of arrival across multiple receivers. By adding additional microphones, the solution matrix can be over-constrained such that send time is not needed. This is a similar trick to that used by GPS, as we described in Section 2.7.2.

Yao's group had previously restricted their empirical studies to centralized systems, as they did not have a platform capable of synchronizing distributed audio sampling with sufficient temporal precision. Indeed, their analysis showed that the spatial precision of their scheme is limited primarily by the variance in the underlying time synchronization, as shown in Figure 9.14.

Recently, using our iPAQ testbed—including RBS time synchronization and the audio server daemon described in Section 9.1.3.2—Wang *et al.* were able to build their first *distributed* beam-forming testbed. Their prototype system was characterized in [WYM02], and later extended to applications such as target classification [WEG03].

9.4   Summary

In this section, we examined three systems that were enabled through the use of Reference-Broadcast Synchronization:

- In our Mote Localization testbed, RBS was used in many ways. Basic RBS was used to synchronize the domain of Berkeley motes with RF
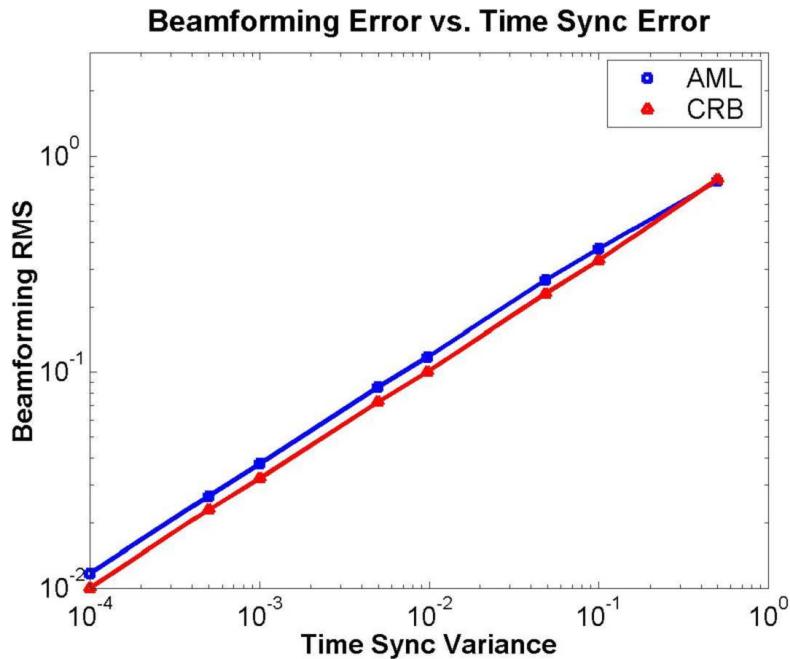
Figure 9.14: An analysis of the effect of time synchronization variance on the expected spatial precision of Yao's beamforming algorithm (AML) and the theoretically optimal result (CRB, or Cramer-Rao Bound). As AML already comes very close to the Cramer-Rao bound, improvements in spatial precision must come from improvements in time synchronization. *Figure courtesy of Hanbiao Wang.*

Monolithics narrowband radios; and, separately, the domain of 802.11-based iPAQ PDAs. The two synchronization domains were linked using a "MoteNIC," combined with an implementation of the RBS multi-hop algorithm. The automatic synchronization-path selection made the sequence of conversions from any mote to any iPAQ transparent to the application developers.

- The SHM system, commercialized by Sensoria Corporation, used acoustic ranging and time synchronization that was conceptually similar to the Mote Localization system. However, this system incorporated localization into a complete, end-to-end application: an autonomous, mobile sensor network that physically reconfigures node positions in response to breaches. This system was a commercial success, as well as demonstrating a successful use of RBS on a third hardware platform—a Hitachi SH4-based host with a hybrid TDMA/FHSS radio.

- Wang *et al.* used our iPAQ testbed to implement the first distributed acoustic beam-forming array, and later extended their work to a target classification application. RBS was an enabling technology for distributing the acoustic sampling data collection; for the lack of appropriate time synchronization, they had previously been constrained to centralized sampling.

Of course, the success of these applications is, in some sense, anecdotal. However, we believe they make a strong case that our time synchronization methods are robust and practical enough to be used in support of real-world applications.

CHAPTER 10

Conclusions and Future Work

The bad news is time flies. The good news is you're the pilot.

—Michael Althsuler

Recent advances in miniaturization and low-cost, low-power design have led to active research in large-scale networks of small, wireless, low-power sensors and actuators. Time synchronization is a critical piece of infrastructure in any distributed system, but wireless sensor networks make particularly extensive use of synchronized time. Almost any form of sensor data fusion or coordinated actuation requires synchronized physical time for reasoning about events in the physical world. However, while the requirements along axes such as tolerable error are often stricter in sensor networks than in traditional distributed systems, energy and channel constraints limit the resources available to meet these goals.

New approaches to time synchronization can better support the broad range of application requirements seen in sensor networks, while meeting the unique resource constraints found in such systems. In our work, we have characterized the unique application requirements in this domain, proposed new synchronization methods that meet those requirements, and successfully implemented and characterized these schemes in the context of several research and commercial systems.

10.1 Contributions

In this section, we review our contributions described earlier in the dissertation. We believe one of the strengths of our work is in its breadth: we have characterized a new problem, described relevant metrics, proposed novel new approaches, implemented and quantitatively characterized our solutions, and seen the implementation through to a number of complete systems—one of which has been commercialized. In the sections that follow, we will review our contributions in each of these areas in more detail.

10.1.1 Characterization of Time in Sensor Networks

An important step toward understanding the problem domain has been the articulation of the relevant metrics. In Chapter 4, we described these metrics and used them to characterize application requirements.

- *Energy utilization*—Some synchronization schemes require extra, energy-hungry equipment (e.g., GPS receivers). Others may have virtually no energy impact (e.g., listening to extant packets already being transmitted for other reasons).

- *Phase and frequency error*—either the dispersion among a group of peers, or maximum error with respect to an external standard. The error tolerance might be as strict as microseconds (e.g., for coherent signal processing on audio signals) or as relaxed as seconds (e.g., to track a slow-moving mobile target).

- *Lifetime*—the duration for which nodes are synchronized. This might be nearly instantaneous (e.g., to compare views of a single event from multiple

vantage points), as long-lived as the network (to track the motion of an object through a sensor field), or persistent forever (e.g., data logging using UTC).

- *Scope and Availability*—the geographic span of nodes that are synchronized, and completeness of coverage within that region. The scope might be as small as a pair of nodes exchanging data, or as large as the entire network.

- *Internal vs. External*—In some applications, the network must be kept internally consistent, but requires no reference to the outside world. In other cases, such as when interacting with people, the network needs a common time reference such as UTC.

- *Cost and Size*—These factors can make a scheme a non-starter. It is unreasonable to put a $50 GPS receiver or a $500 Rubidium oscillator on a node that would otherwise cost $5, or on dust-mote sized nodes.

10.1.2   New Architecture Directions

In some ways, sensor networks are a radical departure from traditional networking; they break many of the assumptions on which prior work is based. In Chapter 5, we saw that these differences have required us to question and modify the design of traditional time synchronization schemes in this new domain. Unlike the Internet, sensor networks must be synchronized with an eye toward energy efficiency; they must often work without infrastructure; timescales of mutually disconnected nodes must be reconciled.

Based on both our own work and the experience of others in this problem space, we have proposed a number of guidelines that are useful in formulating new synchronization schemes for sensor networks:

160

- *Multi-modal*—No *single* synchronization scheme can be optimal on all axes (e.g., precision, lifetime, scope, energy, availability), but most applications do not require peak performance on all axes. A range of schemes should be available to system designers, such that the synchronization that is provided matches what is needed.

- *Tunable*—An ideal synchronization system will minimize its energy use by providing service that is exactly necessary and sufficient for the needs of the application. Tunable parameters can allow synchronization modes to be matched more closely to the requirements of the application.

- *Tiered*—Although Moore's law predicts that hardware for sensor networks will inexorably become smaller, cheaper, and more powerful, technological advances will never prevent the need to make trade-offs. Instead of choosing a single hardware platform that makes a particular set of compromises, we believe an effective design is one that uses a tiered platform consisting of a heterogeneous collection of hardware.

- *Explicit*—Most existing time synchronization schemes make a common assumption: that their goal is to keep the clock synchronized all the time. Applications assume that they can query the clock at any time, and it will be synchronized. We argue that in sensor networks, synchronization should *not* attempt to discipline the phase and frequency of oscillators. A better approach is to let clocks run at their natural rate, and use synchronization to build a set of relationships to other clocks or synthetic timescales. Applications can query the local clock, then perform an *explicit* conversion to another timescale. This has many advantages; for example, it enables post-facto synchronization, peer-to-peer synchronization, and participation in multiple timescales.

161

- *Peer-to-peer*—Traditional synchronization schemes assume the existence of a *global timescale*, usually realized by a *master clock*. We argue that the goal of a synchronization scheme should, instead, be to define the relationship between nearby clocks in a *peer-to-peer* fashion, without necessarily constructing a global timescale. This has a number of benefits—primarily, error between nodes is proportional to the distance between them, rather than their distance from a master clock that is likely to be far away.

10.1.3 Reference-Broadcast Synchronization

In Chapters 6 and 7, we explored a new form of time synchronization, *Reference-Broadcast Synchronization*, that provides more precise, flexible, and resource-efficient network time synchronization than traditional algorithms. The fundamental property of our design is that it *synchronizes a set of receivers with one another*, as opposed to traditional protocols in which senders synchronize with receivers. In addition, we have presented a novel multi-hop algorithm that allows this fundamental property to be maintained across broadcast domains.

In our scheme, nodes periodically send beacon messages to their neighbors using the network's physical-layer broadcast. Recipients use the message's arrival time as a point of reference for comparing their clocks. The message contains no explicit timestamp, nor is it important exactly when it is sent.

RBS has a number of properties that make it attractive. First, by using the broadcast channel to synchronize receivers with one another, the largest sources of nondeterministic latency are removed from the critical path. This results in significantly better-precision synchronization than algorithms that measure round-trip delay. The residual error is often a well-behaved distribution (e.g., Gaussian), allowing further improvement by sending multiple reference broad-

casts. The extra information produces significantly better estimates of relative phase and frequency, and allows graceful degradation in the presence of outliers and lost packets.

We presented a quantitative study that compared an RBS implementation to a carefully tuned installation of GPS-steered NTP. Both used IPAQ PDAs with 802.11 wireless Ethernet. We found that the average synchronization error of RBS was $6.29 \pm 6.45 \mu\text{sec}$, 8 times better than that of NTP in a lightly loaded network. A heavily loaded network further degraded NTP's performance by a factor of 30 but had little effect on RBS. With kernel timestamping hints, RBS achieved an average error of $1.85 \pm 1.28 \mu\text{sec}$, which we expect was limited by the IPAQ's $1\mu\text{sec}$ clock resolution.

Our multihop scheme allows locally coordinated timescales to be federated into a global timescale, across broadcast domains. Precision decays slowly— the average error for an n-hop network is proportional to $\sqrt{n}$. In our test of kernel-assisted RBS across a 4-hop topology, average synchronization error was $3.68 \pm 2.57 \mu\text{sec}$. RBS-federated timescales may also be referenced to an external standard such as UTC if at least one node has access to a source of reference time.

We have implemented RBS on a variety of hardware platforms, where it has proven to be robust and reliable for both performance measurement and in support of real applications. We are confident that these techniques are widely applicable, based on our experience with Berkeley Motes running TinyOS, Linux-based IPAQs, PCs, and Sensoria's WinsNG radios. Each has quirks that have taught us important lessons about RBS.

10.1.4 Post-Facto Synchronization

To save energy in a sensor network, it is a desirable to keep nodes in a low-power state, if not turned off completely, for as long as possible. However, traditional synchronization schemes make the assumption that the clock must be synchronized *before* the event occurs. They must therefore keep clocks synchronized all the time—which is very wasteful if synchronized clocks are needed infrequently and unpredictably.

Using our model of local clocks and explicit conversions (Section 5.2.4), a superior solution falls out quite naturally: *post-facto synchronization*. That is, a node's local clock normally runs *un*synchronized; after an event of interest, a synchronization process is started, allowing timestamps from distributed nodes to be reconciled. Waste is reduced because the system avoids performing synchronization at times when it's not necessary. This scheme works much better than the predictive version: it is much easier to characterize the past than it is to predict the future.

In Chapter 8, we explored two forms of post-facto synchronization. The first, *single-pulse synchronization*, requires frequency calibration at the time the network is deployed (and perhaps periodically afterward). With this information, high-precision retrospective timescales can be quickly constructed using a single packet. Second, we saw that Reference-Broadcast Synchronization can be used for post-facto synchronization. In our test, the phase error of a retrospective RBS timescale was still less than the bit-detection jitter when 60 seconds elapsed between the last synchronization packet and first timestamped event.

10.1.5 Application Experience

As the size and cost sensors and actuators has fallen, it has become feasible to build distributed sensor nodes capable of being embedded in the environment at high density. In Chapter 9, we presented a practical system capable of exploiting that density, achieving 10μsec time synchronization and 10cm spatial localization on a low-cost, low-power, ad-hoc deployable sensor network. This is possible on COTS hardware by making using of novel techniques, including Reference-Broadcast Synchronization and wideband acoustic ranging.

Our group has developed a implemented a centimeter-scale localization service for Berkeley Motes based on acoustic time-of-flight ranging [GBE02]. A set of IPAQs set around the room first establish their own positions within a relatively coordinate system by ranging to one another. Each IPAQ emits an audible "chirp" which has an encoded pseudo-noise sequence [GE01]. IPAQs run a matched filter over their incoming audio data to find the most likely audio sample indicating arrival of the chirp. 802.11-based RBS corrections are then applied to convert this into the equivalent sample number in the sender's timebase, allowing time of flight and therefore range to be computed. Once this startup phase is complete, our "acoustic Motes," specially equipped with small amplifiers and speakers, periodically emit a similar pseudo-noise chirp; IPAQs in the region each compute their ranges to the Mote, and can localize it by trilaterating. In this case, RBS is used to synchronize all Motes in the system to each other. A special "MoteNIC"—a Mote physically attached to an IPAQ—provides translations between the Mote and IPAQ time domains.

Under the auspices of the DARPA SHM program, Sensoria Corporation has developed and commercialized a system that also performs acoustic ranging and multilateration based on RBS time synchronization. Sensoria's system goes be-

yond basic localization, incorporating it into a complete, end-to-end application: an autonomous, mobile sensor network that physically reconfigures node positions in response to breaches. Its success was dramatic, and the system is now being considered by various branches of the armed forces for transition from a DARPA research program into a deployable system.

10.2 Future Work

Although much has been accomplished, there is (as always) much more to be done. In this final section, we describe some areas of interest for future research.

10.2.1 The New Service Model

In Chapter 5, we proposed a significant change in the way that users interact with a time synchronization service. In some cases, the differences are rather dramatic; the idea of discarding a global timescale, for example, is often met with initial skepticism. The old model is very compelling in its simplicity: "What time is it?" is a universally understood interface that matches our experience in the real world. We argued that the new service model enables enormous flexibility, but is it worth the cost in complexity? Or, does the new model fail to go far *enough*, and will it be insufficiently expressive to support some future applications? We do not yet have enough experience to be certain one way or the other.

10.2.2 Reference Broadcast Synchronization

RBS has evolved into a reasonably stable system that has been used outside of our research group. However, there is still much work to be done.

One of the most important issues surrounding the use of RBS is that, so far,

RBS has required customization for each of the applications that have used it. It is not yet clear if a small, standard set of these configurations will be usable across a broad class of applications, or if RBS will always require application-specific binding. The basic method leaves the implementor a great deal of flexibility (or, said less charitably, RBS is underspecified). For example:

- Which nodes are broadcasters? An easily implemented answer is "every node," but this produces far too much redundant information to be worth the impact on network utilization. Real networks need automatic beacon election, but such algorithms may be difficult to generalize across applications. For example, the Sensoria/SHM leader election algorithm used by RBS was really an artifact of the star topology of the network that arose due to the TDMA radios. It is not clear that automatic beacon election can be solved in a way that works across a broad set of applications—each of which may have a unique physical network topology, routing infrastructure, and robustness requirement.

- How is information disseminated? Do nodes compute their own conversion parameters, or (as in the case of our first Mote implementation) send the observations to a larger node that can more easily handle them? Are the conversion parameters broadcast globally to every other node in the network, or is synchronization integrated with routing and applications such that timestamps can be converted hop-by-hop as packets are forwarded? Again, it is not clear that there are "plug-and-play" answers to these questions. As with so many other areas (e.g., routing, data aggregation), interaction with the application is complex. This makes standard services much more difficult to pre-package than in networks such as the Internet that encourage stronger abstractions between layers.

- How does RBS scale? The previous questions have important scaling considerations. For example, a global broadcast of all conversion parameters does not work as the network grows larger. We believe RBS has elegant ways of dealing with scale (e.g., the hop-by-hop conversions), but do not yet have a large-scale implementation as proof.

A number of interesting algorithmic improvements are possible with RBS that we would also like to explore. As the number of beacon nodes per broadcast domain grows, so does the amount of redundant information: can this be leveraged to gain even more accuracy or better rejection of outliers? Recent work by Karp *et al.* has shown some tantalizing results: in an RBS-like scheme, provably *optimal* time synchronization can be achieved by distributing complete information to all neighbors [KEE03]. Further, the authors also suggest that a mostly-localized version of their scheme can achieve *nearly* optimal results. We hope to apply their ideas for collaborative outlier rejection to improve the error bounds in RBS.

Finally, there are a number of quantitative performance questions we would like to explore more fully. For example: precision vs. both the bandwidth used for beacons, and their frequency; minimum time to acquire synchronization to within a given precision bound; post-facto synchronization precision vs. time elapsed from event to reference pulses; and precision with different data filtering algorithms. We would also like to quantify the performance of RBS when used for external (UTC) synchronization over multiple hops, vs. using NTP in the same topology.

10.2.3 Post-Facto Synchronization

The most important open question surrounding post-facto synchronization (PFS) is a simple one: is it worthwhile? If synchronization is needed occasionally and

unpredictably, it seems that PFS should be a clear choice. However, to date, we have not seen an application where the energy saved by turning timesync off is worth the resulting increase in system complexity.

It is possible that this is simply an artifact of our field's immaturity. Most of our work so far has been in the context of prototype or demonstration systems. We seem just on the verge of the next generation: deployments of the first long-lived, in-situ systems where energy-efficiency is of a practical rather than academic concern. It could be that when we reach that point, PFS will become quite compelling. Or, it could be that even in regimes where maximum efficiency is required, there is always *some* background radio traffic, and the incremental benefit of stopping timesync is not worthwhile.

Part of the cost/benefit analysis is the loss in precision: as greater times elapse between an event and the start of PFS, the precision decays. In applications that require the utmost precision, PFS may never be viable. In answering this side of the question, deeper performance analysis of PFS is needed. Our experiments with it so far have been a proof-of-concept, but we have yet to perform a systematic exploration of the effects of longer silence intervals on the accuracy and variance of synchronization.

10.2.4 Applications

At its heart, time synchronization is mostly a means to an end. It has little significance without applications that need it. We look forward to continuing our collaborations with colleagues in the sensor network community who have research interests that require precise time synchronization: distributed signal processing, acoustic ranging, object tracking, coordinated actuation, and so forth. Our existing application work has been an enormous milestone, but represents only

demonstration systems: they have been deployed for, at most, a few days. Our ambition is to see an implementation of our techniques through to a fully-fielded, operational system—one that goes beyond only an embodiment of systems research, and produces real scientific results.

Two such systems are in development: a 100-node tiered-architecture microclimate array, and a 50-node collaborative multi-hop seismic array. We are working with our partners in the natural sciences to create a system that is both scientifically useful and advances the state of the art in sensor system design.

References

[AAH97]   David W. Allan, Neil Ashby, and Clifford C. Hodge. "The Science of Timekeeping." Technical report, Hewlett Packard Application Note 1287, 1997.
http://www.allanstime.com/Publications/DWA/Science_Timekeeping.

[ADL98]   G. Asada, M. Dong, T.S. Lin, F. Newberg, G. Pottie, W.J. Kaiser, and H.O. Marcy. "Wireless Integrated Network Sensors: Low Power Systems on a Chip." In *Proceedings of the European Solid State Circuits Conference*, 1998.

[AG01]   Claude Audoin and Bernard Guinot. *The Measurement of Time: Time, Frequency, and the Atomic Clock.* Cambridge University Press, 2001.

[APK00]   A.A. Abidi, G.J. Pottie, and W.J. Kaiser. "Power-conscious design of wireless circuits and systems." *Proceedings of the IEEE*, **88**(10):1528–45, October 2000.

[ASB96]   Bülent Abali, Craig B. Stunkel, and Caroline Benveniste. "Clock Synchronization on a Multicomputer." Technical Report 8028, IBM Research Division, T.J. Watson Research Center, 1996.

[ASS02]   I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. "Wireless Sensor Networks: A Survey." *Computer Networks*, **38**(4):393–422, March 2002.

[BCY98]   A.A. Berlin, J.G. Chase, M.H. Yim, B.J. Maclean, M. Oliver, and S.C. Jacobsen. "MEMS-Based Control of Structural Dynamic Instability." *Journal of Intelligent Material Systems and Structures*, **9**(7):574–586, 1998.

[BD82]   Ian R. Bartky and Steven J. Dick. "The First North American Time Ball." *Journal for the History of Astronomy*, **13**:50–54, 1982.

[Bee81]   R.E. Beehler. "Time/Frequency Services of the U.S. National Bureau of Standards and Some Alternatives for Future Improvement." *Journal of Electronics and Telecommunications Engineers*, **27**:389–402, Jan 1981.

[BGS00]   Philippe Bonnet, Johannes E. Gehrke, and Praveen Seshadri. "Querying the Physical World." *IEEE Personal Communications*, **7**(5):10–15, October 2000.

[BHE03]   Nirupama Bulusu, John Heidemann, Deborah Estrin, and Tommy
           Tran. "Self-configuring Localization Systems: Design and
           Experimental Evaluation." *ACM Transactions on Embedded
           Computing Systems*, p. To appear., May 2003.

[BLU]   "Bluetooth SIG." www.bluetooth.org.

[CE02]   Alberto Cerpa and Deborah Estrin. "ASCENT: Adaptive
           Self-Configuring sEnsor Networks Topologies." In *Proceedings of the
           Twenty First Annual Joint Conference of the IEEE Computer and
           Communications Societies (INFOCOM 2002)*, New York, NY, USA,
           June 23–27 2002. IEEE.

[CEE01]   Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael
           Hamilton, and Jerry Zhao. "Habitat Monitoring: Application Driver
           for Wireless Communications Technology." In *Proceedings of the
           2001 ACM SIGCOMM Workshop on Data Communications in Latin
           America and the Caribbean*, April 2001. Available at
           http://www.isi.edu/scadds/papers/CostaRica-oct01-final.ps.

[CNS]   CNS Systems, Inc. "The CNS Clock."
           http://www.cnssys.com/cnsclock.

[Cri89]   Flaviu Cristian. "Probabilistic clock synchronization." *Distributed
           Computing*, **3**:146–158, 1989.

[DGT96]   Donald T. Davis, Daniel E. Geer, and Theodore Ts'o. "Kerberos
           with Clocks Adrift: History, Protocols, and Implementation."
           Computing Systems, **9**(1):29–46, Winter 1996.

[EBB03]   Jeremy Elson, Solomon Bien, Naim Busek, Vladimir Bychkovskiy,
           Alberto Cerpa, Deepak Ganesan, Lewis Girod, Ben Greenstein, Tom
           Schoellhammer, Thanos Stathopoulos, and Deborah Estrin.
           "EmStar: An Environment for Developing Wireless Embedded
           Systems Software." Technical Report 0009, Center for Embedded
           Networked Sensing, University of California, Los Angeles, March
           2003. http://lecs.cs.ucla.edu/publications.

[EE01]   Jeremy Elson and Deborah Estrin. "Time Synchronization for
           Wireless Sensor Networks." In *Proceedings of the 15th International
           Parallel and Distributed Processing Symposium (IPDPS-01)*. IEEE
           Computer Society, April 23–27 2001.

[EG02]   Laurent Eschenauer and Virgil D. Gligor. "A key-management
           scheme for distributed sensor networks." In Vijay Atlury, editor,
           *Proceedings of the 9th ACM Conference on Computer and
           Communication Security (CCS-02)*, pp. 41–47, New York, November
           18–22 2002. ACM Press.

[EGH99]   Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish
           Kumar. "Next Century Challenges: Scalable Coordination in Sensor
           Networks." In *Proceedings of the fifth annual ACM/IEEE
           international conference on Mobile computing and networking*, pp.
           263–270, 1999.

[ER02]   Jeremy Elson and Kay Römer. "Wireless Sensor Networks: A new
           regime for time synchronization." In *Proceedings of the First
           Workshop on Hot Topics In Networks (HotNets-I)*, Princeton, New
           Jersey, October 2002.

[FAM]   "Familiar Linux." http://www.handhelds.org.

[GBE02]   Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah
           Estrin. "Locating tiny sensors in time and space: A case study." In
           *In Proceedings of the International Conference on Computer Design
           (ICCD 2002)*, Freiburg, Germany, September 2002.
           http://lecs.cs.ucla.edu/Publications.

[GE01]   Lewis Girod and Deborah Estrin. "Robust Range Estimation Using
           Acoustic and Multimodal Sensing." In *Proceedings of the IEEE/RSJ
           International Conference on Intelligent Robots and Systems (IROS
           2001)*, March 2001.

[Gif79]   D. K. Gifford. "Weighted Voting for Replicated Data." In
           *Proceedings of the 7th Symposium on Operating Systems Principles
           (7th SOSP'79), Operating Systems Review*, pp. 150–161, Pacific
           Grove, California, December 1979. ACM, New York, USA.

[GKA02]   Saurabh Ganeriwal, Ram Kumar, Sachin Adlakha, and Mani B.
           Srivastava. "Network-wide Time Synchrinization in Sensor
           Networks." Technical report, University of California, Dept. of
           Electrical Engineering, 2002.

[GZ89]   R. Gusell and S. Zatti. "The accuracy of clock synchronization
           achieved by TEMPO in Berkeley UNIX 4.3 BSD." *IEEE
           Transactions on Software Engineering*, **15**:847–853, 1989.

[HC01]   Jason Hill and David Culler. "A wireless embedded sensor
           architecture for system-level optimization." Technical report, U.C.
           Berkeley, 2001.

[HSI01]   John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh
           Govindan, Deborah Estrin, and Deepak Ganesan. "Building Efficient
           Wireless Sensor Networks with Low-Level Naming." In *Proceedings
           of the Symposium on Operating Systems Principles*, pp. 146–159,
           Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM.

[HSW00]   Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler,
           and Kristofer Pister. "System architecture directions for networked
           sensors." In *Proceedings of the Ninth International Conference on
           Architectural Support for Programming Languages and Operating
           Systems (ASPLOS-IX)*, pp. 93–104, Cambridge, MA, USA,
           November 2000. ACM.

[IGE00]   Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin.
           "Directed Diffusion: A Scalable and Robust Communication
           Paradigm for Sensor Networks." In *Proceedings of the Sixth Annual
           International Conference on Mobile Computing and Networking*, pp.
           56–67, Boston, MA, August 2000. ACM Press.

[ISO99]   ISO/IEC. "IEEE 802.11 Standard." IEEE Standard for Information
           Technology, ISO/IEC 8802-11:1999(E), 1999.

[Kap96]   Elliott D. Kaplan, editor. *Understanding GPS: Principles and
           Applications.* Artech House, 1996.

[KEE03]   Richard Karp, Jeremy Elson, Deborah Estrin, and Scott Shenker.
           "Optimal and Global Time Synchronization in Sensornets."
           Technical Report 0009, Center for Embedded Networked Sensing,
           University of California, Los Angeles, April 2003.
           http://lecs.cs.ucla.edu/Publications.

[KKP99]   J.M. Kahn, R.H. Katz, and K.S.J. Pister. "Next century challenges:
           mobile networking for Smart Dust." In *Proceedings of the fifth
           annual ACM/IEEE international conference on Mobile computing
           and networking*, pp. 271–278, 1999.

[KPH03]   Juyoul Kim, Yeonjeong Park, and Thomas C. Harmon. "Real-Time
           Model Parameter Estimation for Analyzing Transport in Porous
           Media." Technical report, Center for Embedded Networked Sensing,

University of California, Los Angeles, May 2003.
http://www.cens.ucla.edu/Project-Descriptions/Contam.

[KS89]   H. Kopetz and W. Schwabl. "Global Time in Distributed Real-Time
             Systems." Technical Report 15/89, Technische Universität Wien,
             1989.

[Lam78]   Leslie Lamport. "Time, clocks, and the ordering of events in a
             distributed system." *Communications of the ACM*, **21**(7):558–65,
             1978.

[Lan83]   David S. Landes. *Revolution in Time: Clocks and the Making of the
             Modern World*. Belknap Press, 1983.

[LEH03]   Y.W. Law, S. Etalle, and P.H. Hartel. "Assessing Security-Critical
             Energy-Efficient Sensor Networks." In *Proceedings of IFIP WG 11.2
             Small Systems Security Conference*, Athens, Greece, May 2003.

[LL99]   Kristine Larson and Judah Levine. "Time Transfer using the phase
             of the GPS Carrier." *IEEE Transactions On Ultrasonics,
             Ferroelectrics and Frequency Control*, **46**:1001–1012, 1999.

[LMC99]   Cheng Liao, Margaret Martonosi, and Douglas W. Clark.
             "Experience with an Adaptive Globally-Synchronizing Clock
             Algorithm." In *Eleventh Annual ACM Symposium on Parallel
             Algorithms and Architectures (SPAA '99)*, pp. 106–114, New York,
             June 1999.

[LS96]   Henrik Lönn and Rolf Snedsbøl. "Efficient Synchronisation, Atomic
             Broadcast and Membership Agreement in a TDMA protocol."
             Technical report, CTH, Dept. of Computer Engineering, Laboratory
             for Dependable Computing (LDC), 1996.

[MB83]   R. M. Metcalfe and D. R. Boggs. "Ethernet: Distributed Packet
             Switching for Local Computer Networks." *Communications of the
             ACM*, **26**(1):90–95, January 1983.

[MF02]   Samuel Madden and Michael J. Franklin. "Fjording the Stream: An
             Architecture for Queries Over Streaming Sensor Data." In
             *Proceedings of the 18th International Conference on Data
             Engineering (ICDE'02)*, San Jose, California, February 2002.

[MFH02]   Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and
             Wei Hong. "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor

Networks." In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 131–146, Boston, MA, USA, December 2002.

[MFN00]   M. Mock, R. Frings, E. Nett, and S. Trikaliotis. "Continuous Clock Synchronization in Wireless Real-Time Applications." In *The 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pp. 125–133, Washington - Brussels - Tokyo, October 2000. IEEE.

[MGE02]   William Merrill, Lewis Girod, Jeremy Elson, Kathy Sohrabi, Fredric Newberg, and William Kaiser. "Autonomous Position Location in Distributed, Embedded, Wireless Systems." In *Proceedings of IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, CA, September 2002. http://www.sensoria.com/publications.html.

[Mil94a]   David L. Mills. "Internet Time Synchronization: The Network Time Protocol." In Zhonghua Yang and T. Anthony Marsland, editors, *Global States and Time in Distributed Systems*. IEEE Computer Society Press, 1994.

[Mil94b]   David L. Mills. "Precision synchronization of cmputer network clocks." *ACM Computer Comm. Review*, **24**(2):28–43, April 1994.

[Mil98]   David L. Mills. "Adaptive hybrid clock discipline algorithm for the network time protocol." *IEEE/ACM Transactions on Networking*, **6**(5):505–514, October 1998.

[Mil03]   David L. Mills. "A brief history of NTP time: confessions of an Internet timekeeper." *ACM Computer Communications Review*, **33**(3), July 2003.

[MJ93]   Steven McCanne and Van Jacobson. "The BSD Packet Filter: A New Architecture for User-level Packet Capture." In USENIX Association, editor, *Proceedings of the Winter 1993 USENIX Conference: January 25–29, 1993, San Diego, California, USA*, pp. 259–269, Berkeley, CA, USA, Winter 1993. USENIX.

[MK00]   David L. Mills and P.-H. Kamp. "The nanokernel." In *Proceedings of the Precise Time and Time Interval (PTTI) Applications and Planning Meeting*, Reston, VA, November 2000.

[MKM99]   J. Mannermaa, K. Kalliomaki, T. Mansten, and S. Turunen. "Timing performance of various GPS receivers." In *Proceedings of*

the 1999 Joint Meeting of the European Frequency and Time Forum and the IEEE International Frequency Control Symposium, pp. 287–290, April 1999.

[MMK99]   D.N. Matsakis, M. Miranian, and P. Koppang. "Steering The U.S. Naval Observatory (USNO) Master Clock." In *Proceedings of the Institute of Navigation National Technical Meeting: Vision 2010, Present and Future*, January 1999.

[MO85]   Keith Marzullo and Susan Owicki. "Maintaining Time in a Distributed Server." *Operating Systems Review*, **19**(3):44–54, July 1985.

[MPS02]   Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. "Wireless Sensor Networks for Habitat Monitoring." In *Proceedings of the First ACM Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, USA, September 28 2002.

[NES]   DARPA Information Processing Technology Office (IPTO). "NEST: Networked Embedded Software Technology." http://www.darpa.mil/ipto/Solicitations/CBD_01-06.html.

[NPL02]   NIST National Physics Laboratory. "A Walk Through Time." http://physics.nist.gov/time, 2002.

[PK00]   Gregory J. Pottie and William J. Kaiser. "Wireless Integrated Network Sensors." *Communications of the ACM*, **43**(5):51–58, May 2000.

[PSW01]   Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. "SPINS: Security Suite for Sensor Networks." In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking (MOBICOM-01)*, pp. 189–199, New York, July 16–21 2001. ACM Press.

[PV02]   Attila Pásztor and Darryl Veitch. "PC Based Precision Timing Without GPS." In *Proceedings of ACM SIGMETRICS 2002*, Marina del Rey, California, June 15–19 2002. International Conference on Measurement and Modeling of Computer Systems.

[R01]   Kay Römer. "Time Synchronization in Ad Hoc Networks." In *Proceedings of MobiHoc 2001*, Long Beach, CA, Oct 2001.

[RSS03]   Mohammad Rahimi, Hardik Shah, Gaurav Sukhatme, John
          Heidemann, and Deborah Estrin. "Energy harvesting in mobile
          sensor networks." Technical report, Center for Embedded Networked
          Sensing, 2003. http://www.cens.ucla.edu/Project-
          Descriptions/energyharvesting/EnergyHarvesting.pdf.

[Rub79]   Izhak Rubin. "Message Delays in FDMA and TDMA
          Communication Channels." *IEEE Trans. Communin.*,
          COM27(5):769–777, May 1979.

[SA98]   Dava Sobel and William J.H. Andrewes. *The Illustrated Longitude.*
          Walker & Co., 1998.

[SA02]   Weilian Su and Ian F. Akyildiz. "Time-Diffusion Synchronization
          Protocol for Sensor Networks." Technical report, Georgia Institute of
          Technology, Broadband and Wireless Networking Laboratory, 2002.

[SHM]   DARPA Advanced Technology Office (ATO). "Self-Healing
          Minefield." http://www.darpa.mil/ato/programs/SHM/.

[SME02]   Katayoun Sohrabi, William Merrill, Jeremy Elson, Lewis Girod,
          Fredric Newberg, and William Kaiser. "Autonomous Position
          Location in Distributed, Embedded, Wireless Systems." In
          *Proceedings of IEEE CAS Workshop on Wireless Communications
          and Networking*, Pasadena, CA, September 2002.
          http://www.sensoria.com/publications.html.

[Soh00]   Katayoun Sohrabi. *On Low Power Self Organizing Sensor Networks.*
          PhD thesis, University of California, Los Angeles, 2000.

[SPK00]   W. Schaefer, A. Pawlitzki, and T. Kuhn. "New Trends in Two-Way
          Time and Frequency Transfer via Satellite." In *Proceedings of the
          31st Annual Precise Time and Time Interval (PTTI) Systems and
          Applications Meeting*, 2000.

[SRS02]   Gabriel T. Sibley, Mohammad H. Rahimi, and Gaurav S. Sukhatme.
          "Robomote: A Tiny Mobile Robot Platform for Large-Scale Sensor
          Networks." In *Proceedings of the IEEE International Conference on
          Robotics and Automation (ICRA2002)*, 2002.

[ST87]   T. K. Srikanth and Sam Toueg. "Optimal Clock Synchronization."
          J-ACM, **34**(3):626–645, July 1987.

[Tho97]   C. Thomas. "The accuracy of the international atomic timescale TAI." In *Proceedings of the 11th European Frequency and Time Forum*, pp. 283–289, 1997.

[Vig92]   John R. Vig. "Introduction to Quartz Frequency Standards." Technical Report SLCET-TR-92-1, Army Research Laboratory, Electronics and Power Sources Directorate, October 1992. Available at http://www.ieee-uffc.org/freqcontrol/quartz/vig/vigtoc.htm.

[VR92]   Paulo Veríssimo and Luis Rodrigues. "A Posteriori Agreement for Fault-Tolerant Clock Synchronization on Broadcast Networks." In Dhiraj K. Pradhan, editor, *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS '92)*, p. 85, Boston, MA, July 1992. IEEE Computer Society Press.

[VRC97]   Paulo Veríssimo, Luis Rodrigues, and Antonio Casimiro. "CesiumSpray: a Precise and Accurate Global Time Service for Large-scale Systems." Tech. Rep. NAV-TR-97-0001, Universidade de Lisboa, 1997.

[WC02]   Kamin Whitehouse and David Culler. "Calibration as parameter estimation in sensor networks." In *Proceedings of the first ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 59–67, 2002.

[WEG03]   Hanbiao Wang, Jeremy Elson, Lewis Girod, Deborah Estrin, and Kung Yao. "Target Classification and Localization in Habitat Monitoring." In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2003)*, Hong Kong, China, April 2003.

[WHF92]   Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. "The active badge location system." *ACM Transactions on Information Systems*, **10**(1):99–102, 1992.

[WJH97]   A. Ward, A. Jones, and A. Hopper. "A new location technique for the active office." *IEEE Personal Communications*, **4**(5):42–47, October 1997.

[WL98]   Jay Werb and Colin Lanzl. "Designing a positioning system for finding things and people indoors." *IEEE Spectrum*, **35**(9):71–78, September 1998.

[WYM02] H. Wang, L. Yip, D. Maniezzo, J.C. Chen, R.E. Hudson, J.Elson, and K.Yao. "A Wireless Time-Synchronized COTS Sensor Platform Part II–Applications to Beamforming." In *Proceedings of IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, CA, September 2002. http://lecs.cs.ucla.edu/Publications.

[YHR98] K. Yao, R.E. Hudson, C.W. Reed, D. Chen, and F. Lorenzelli. "Blind beamforming on a randomly distributed sensor array system." *IEEE Journal of Selected Areas in Communications*, **16**(8):1555–1567, Oct 1998.